THE UNIVERSITY *of* EDINBURGH

Spatio-temporal Logic for the Analysis of Biochemical Models

Christopher Jon Banks

Doctor of Philosophy

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2015

Abstract

Process algebra, formal specification, and model checking are all well studied techniques in the analysis of concurrent computer systems. More recently these techniques have been applied to the analysis of biochemical systems which, at an abstract level, have similar patterns of behaviour to concurrent processes. Process algebraic models and temporal logic specifications, along with their associated model-checking techniques, have been used to analyse biochemical systems.

In this thesis we develop a spatio-temporal logic, the Logic of Behaviour in Context ($\mathcal{LBC}$), for the analysis of biochemical models. That is, we define and study the application of a formal specification language which not only expresses temporal properties of biochemical models, but expresses spatial or contextual properties as well. The logic can be used to express, or specify, the behaviour of a model when it is placed into the context of another model.

We also explore the types of properties which can be expressed in $\mathcal{LBC}$, various algorithms for model checking $\mathcal{LBC}$—each an improvement on the last, the implementation of the computational tools to support model checking $\mathcal{LBC}$, and a case study on the analysis of models of post-translational biochemical oscillators using $\mathcal{LBC}$.

We show that a number of interesting and useful properties can be expressed in $\mathcal{LBC}$ and that it is possible to express highly useful properties of real models in the biochemistry domain, with practical application. Statements in $\mathcal{LBC}$ can be thought of as expressing *computational experiments* which can be performed automatically by means of the model checker. Indeed, many of these computational experiments can be *higher-order* meaning that one succinct and precise specification in $\mathcal{LBC}$ can represent a number of experiments which can be automatically executed by the model checker.

Lay Summary

Process algebraic languages provide a way to describe process models in an unambiguous and machine-readable manner. These languages have been applied to the construction of biochemical process models, that is models of the dynamics of interacting biochemical species, proteins, or other reagents. The use of process algebra for modelling these processes allows for a model description which is amenable to more automatic manipulation than a classical mathematical model or a model written in a general purpose programming language.

The use of process algebra also allows for the use of a number of associated tools. One important tool is that of logical specification and model checking. Logical specification allows the modeller to write precise and unambiguous statements about the model and its behaviour, statements which are hopefully true of the model. Model checking is the automatic means of checking a whether a logical statement is true of a model.

In this thesis we develop the Logic of Behaviour in Context ($\mathcal{LBC}$) which is a logical specification language designed with the properties of biochemical processes in mind. $\mathcal{LBC}$ can be used to describe both temporal and contextual properties. Temporal properties describe a model's state over time and contextual properties describe a model's state in some context. By context, here, we mean environmental context, like if our process interacts with another process or if some new species or reagent is added to the process.

We also develop an efficient model-checking procedure for the language and implement it in a software tool. The interplay between temporal and contextual properties allows the modeller to, in effect, define "computational experiments" on a model. The automated model-checking software allows these experiments to be executed.

Finally we design a number of these computational experiments as part of an interdisciplinary case study into real biochemical models. The results of the study show that $\mathcal{LBC}$ is a practical and efficient tool for experimenting with biochemical process models.

Acknowledgements

Foremost, I would like to thank my supervisors Ian Stark and Jane Hillston for their advice and support. I am greatly indebted to them both, but especially to Ian for inspiring me and keeping me on track continuously. I would also like to thank a number of people who have helped me significantly along the way: Marek Kwiatkowski, Sylvain Soliman, and Luca Bortolussi.

Special thanks goes to Daniel Seaton for working with me on the case study and for explaining the biology of circadian clocks to me.

I must also thank the various inhabitants of IF3.50 for keeping me entertained, intellectually stimulated, and sane, on a daily basis: Ohad Kammar, Dimitris Milios, Alireza Pourranjbar, Krzysztof Gorgolewski, Ondrej Mandula, Chris Ball, Anastasis Georgoulas, Yota Katsikouli, Ludovica Louisa Vissat, and Cheng Feng.

I have been continuously aided and encouraged by the members of the PEPA group who do not appear above: Stephen Gilmore, Vashti Galpin, Allan Clark, and Maria Luisa Guerriero.

I would also like to thank my examiners Stephen Gilmore, François Fages, and Andrzej Kierzek for a smooth examination, but leaving me with plenty to think about, and Tom Michoel for supporting me as a research fellow whilst I finished this thesis.

And finally, I would not have survived without the wonderful and beautiful intellectual environment provided by the Laboratory for Foundations of Computer Science and the fine set of people who work within it.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification.

Chapters 4 and 5 are refined and expanded versions of previously published material [12]. Some material has also been published in a workshop paper [11]. Other material in the thesis is being prepared for publication.

(Christopher Jon Banks)

To Catherine, for everything.

Table of Contents

List of Figures

Chapter 1

Introduction

In this thesis we develop a framework for the formal modelling, specification, and verification of biochemical reaction networks. We draw from the existing body of work in computer science on process algebra, temporal logic, and model checking to develop a system which allows a modeller to perform "computational experiments" on biochemical models.

In the early 2000s Regev and Shapiro published their seminal paper [80] on the abstraction of biochemical processes as models in process algebra. Since then a plethora of process algebraic formalisms have been devised, both to address the needs of modelling specific types of biochemical process or to generalise the approach to provide a platform for the formal modelling of all kinds of processes.

The use of formalisms drawn from computer science meant that an existing wealth of supporting tools and techniques could be applied to this new problem domain. Especially useful were the techniques of formal specification and the associated technique and toolset of model checking. Likewise, development followed on the use of formal specification and model-checking tools designed to handle the specifics of biochemical processes and their behaviour.

1.1 Motivation

The motivation for this thesis arises, foremost, from the need to address one of these specific properties of biochemical processes and their behaviour. That is, in

the study of biochemical processes it is often the case that the process warrants study in varying environmental conditions (*context*). It is also the case that, as in *wet lab* biochemistry, one might wish to perform experiments by introducing new biochemical species to the process under investigation (*introduction*).

In this thesis we develop the foundations of a language for formally expressing properties of biochemical systems, including the expression of properties where behaviour arises given some environmental context or the introduction of something new to a process. It is our aim to provide a precise language with which to describe these properties, statements which can also be seen as a specification of *formal experiments*; that is, a precise means to describe the desired behaviour of a process under some environmental condition or perturbation.

It is greatly desirable—given that the motivation for this endeavour is largely practical and we wish to enable real-world application in biochemical modelling—that the developed languages and model-checking tools be implemented efficiently. To enable this, efficient model-checking algorithms need to be developed.

A final motivation is the desire to further develop the capabilities underlying the modelling language. This thesis builds on the Continuous π-calculus (cπ), a continuous space process algebra, the work of Kwiatkowski and Stark [62, 63, 64]. The idea originally arose as a means of extending cπ with a specification logic and model checker. It was particularly desired to make use of compositionality in the logic—indeed the notion of context and introduction arose from this.

1.2 Contributions

In this thesis we address the above criteria and develop a logical framework and tool platform for expressing and checking properties of biochemical models, especially those which involve context or introduction. The main contributions of this thesis are, in summary:

- the $\mathcal{LBC}$ language definition and formal semantics;
- a number of model-checking algorithms for $\mathcal{LBC}$;
- an interdisciplinary case-study on the analysis of post-translational biochemical oscillator models using $\mathcal{LBC}$; and

- working implementations of the algorithms, a systematic testing and profiling suite, and a user interface.

The first contribution is the definition and formal semantics of the Logic of Behaviour in Context ($\mathcal{LBC}$). The logic extends standard temporal logic with real-valued constraints and with a *context modality* which specifies behaviour in the context of another process. We present this in the established style for the semantics of temporal logics—using the definition of a satisfaction relation. We present two alternative semantics—absolute time and relative time—both of which have their *raisons d'être*: the absolute-time semantics is more straightforward to model check, however the relative-time semantics is much more useful in practice. We conclude the chapter by discussing the types of property which are expressible in the logic and describe how $\mathcal{LBC}$ statements can be seen as *precise and succinct descriptions of computational experiments* on models.

The next contribution is a series of model-checking algorithms for $\mathcal{LBC}$. The first three algorithms are based on approximate model checking over discrete simulation traces of a model. We begin with a naive algorithm which is defined succinctly as a direct mapping from the semantics; this algorithm, however, is very costly. We continue by defining a less costly algorithm by using a dynamic programming approach. We then complete the set of trace-based algorithms by defining a hybrid algorithm, taking elements of the two previous algorithms; this algorithm allows the model checker to terminate early—without traversing the whole simulation trace—if, say, the witness for a property is found, thus again improving the cost of model checking. However, these two improved trace-based algorithms only support the absolute time semantics.

We then move to basing the model-checking algorithm on Boolean signals. The signal-based algorithm allows the relative-time semantics which is more useful in practice. Signal-based checking also improves the cost of checking the temporal fragment of the logic. However, the cost of checking a context modality remains the same.

Of course, to achieve both the raw performance results and the applied case study results we required an implementation. We built a suite of tools for building models in cπ specifications in $\mathcal{LBC}$ and implementations of each of the model-checking algorithms. We present some details of the tool; this also serves to

enable a third party to replicate the results of this thesis.

To evaluate the usefulness of $\mathcal{LBC}$ for expressing properties of real biochemical models and for its use in analysing such models we undertook a case study. The case study focuses on the modelling of a biochemical post-translational oscillator and its analysis using $\mathcal{LBC}$ and model checking. We show that complex properties of the model can be expressed in the logic. Furthermore, we show that the logic can be used to specify *computational experiments* on the model and these experiments can be *higher-order*; that is, the logic expresses succinctly a property which might require many "standard" computational experiments to verify and the model checker automatically does the work of verifying the property.

Model checking the context modality—using any of the previous algorithms—is relatively costly. A context modality nested within a temporal modality forces the model to be re-compiled and its dynamics re-computed an arbitrarily large number of times. Our final contribution is to develop the foundations of a method to reduce the cost of checking the context modality, using techniques from sensitivity analysis.

1.3 Thesis structure

Chapter 1 is this introduction.

Chapter 2 contains a survey of necessary background literature: an introduction to process algebra, temporal logic, and model checking. It also surveys the relevant literature upon which this thesis is built, as well as related work in the field.

Chapter 3 introduces the Continuous π-calculus. The language presented is that of Kwiatkowski and Stark, following Kwiatkowski's thesis [62], but with a refined presentation for this thesis. We follow this by presenting a simple, but motivating example of the use of the language for describing a simple biochemical process.

Chapter 4 introduces our novel formal specification language for biochemical processes with context: the Logic of Behaviour in Context ($\mathcal{LBC}$). We present the syntax and semantics in an established style by defining a sat-

isfaction relation. We then present a systematic survey of the kinds of property which can be expressed using the language; this survey serves to give the reader logical examples and an intuition for their meaning.

Chapter 5 develops a first set of model-checking algorithms for $\mathcal{LBC}$, based on analysing traces. We present three approaches, each building on and improving upon its predecessor. We show the runtime performance of the implementation of each algorithm and compare this to the predicted complexity. Finally we describe the limitations of using a trace-based algorithm.

Chapter 6 develops the second approach to model checking, based on boolean signals. Again we show the runtime performance of the algorithm in implementation and we compare this with the performance of the previous algorithms. Finally we conclude and outline some open problems.

Chapter 7 details the substantial modelling case study undertaken with the aim of evaluating the use of $\mathcal{LBC}$. We begin by motivating the case study and presenting its background. We then show how the biochemical process under investigation can be modelled using cπ and show some of its basic properties using the cπ analysis tools. We then show how $\mathcal{LBC}$ can be used to further elucidate some of the more complex properties of the model, perform computational experiments, and to test conjectures about the model which were not feasible to test using the standard analyses. Finally we conclude by evaluating $\mathcal{LBC}$'s efficacy in this application and its general properties.

Chapter 8 provides some details of the software tools which have been developed to support this thesis.

Chapter 9 presents a foundation for addressing some of the open problems described in previous chapters. We describe how, using a technique which draws from sensitivity analysis, we have a possible means to address the complexity limit we have seen in previous chapters.

Chapter 10 concludes the thesis by evaluating the results and proposing some further work.

Chapter 2

Background and literature

In this chapter we introduce some of the necessary background and literature relevant to the work contained within this thesis. We also introduce some related work. Topics covered in this chapter include: process algebra, temporal logic, spatial logic, model checking, and the use of these formalisms for quantitative specification and modelling.

2.1 Formal quantitative modelling languages

Formal quantitative modelling languages arose from the need to have unambiguous (machine readable) and concise representations of physical systems. The sorts of systems modelled are any systems where, at an abstract level, objects or agents interact in some way; this ranges from computer systems and protocols, to biological systems, to traffic control systems, and beyond. Indeed the specific sort of formal models we consider in this thesis—process algebras—arose from the study of complex concurrent and distributed computing systems.

2.1.1 Process algebra

Process algebras began life as a means of denoting models of concurrent computation. A vast number of process algebraic languages exist, each having been designed with a slightly different goal in mind. However, each has a common set of elements: agents, actions, sequential composition, parallel composition,

and some notion of which actions are shared between agents. A typical process algebraic language may look something like the following:

We have a set of agents—where agents are denoted A, B, etc.—and a set of actions—where actions are denoted m, $\hat{m}$, etc. We define m and $\hat{m}$ to be action and co-action, respectively. If one agent performs an action and another performs a co-action at the same time then they cooperate over these actions. For example, m could be the sending of a message and $\hat{m}$ could be the receipt of a message. There is also usually an inactive agent 0 which can represent termination or deadlock.

We then have two operators: sequential composition ; and parallel composition $\|$. Using all of the above we can now define two agents:

$$A = m\ ;\ 0$$

$$B = \hat{m}\ ;\ B$$

Agent A first sends a message (m) and then terminates (0). Agent B receives a message ($\hat{m}$) and then becomes B again, awaiting receipt of another message. We can then define a message-passing system S:

$$S = A \parallel B$$

which will evolve as follows, where $\rightarrow$ represents a transition from one state to the next:

$$\begin{aligned} S = A \parallel B &= m\ ;\ 0 \parallel \hat{m}\ ;\ B \\ &\rightarrow 0 \parallel B \end{aligned}$$

This very simple system only has one transition, of course: the sending of a message from A to B. Once this shared action has been performed, no other action is possible because A is now inactive (0) and there is no send (m) action to complement B's receive ($\hat{m}$) action. Although this example is trivial, one can see that much more complicated behaviour can be built from such components.

One of the first widely recognised and significant process algebra was Milner's Calculus of Communicating Systems (CCS) [68, 69], which established the language, its semantics, and various notions of process equivalence—notably bisimulation. A number of alternative approaches were developed independently, such

as Communicating Sequential Processes (CSP) [54], and Algebra of Communicating Processes (ACP) [14]. A review paper by Baeten [8] presents a good overview of these foundational process algebras.

Inevitably a number of extended process algebras have since been proposed, each addressing aspects of systems which are difficult or impossible to model in the foundational process algebras. Some notable examples are: the π-calculus [70] for mobile processes where network topology can change, Performance Evaluation Process Algebra (PEPA) [53] and the Stochastic π-calculus [75] for performance evaluation of concurrent processes. The π-calculus extends the basic algebra with a means to pass not only messages, but also to pass *channels* along which the receiving agent can pass a message; this is a mechanism introduced to model the reconfigurable topology of mobile networks. PEPA allows the passage of messages to have an associated stochastic rate, thereby allowing the analysis of the runtime performance of systems. The Stochastic π-calculus combines both of these aspects.

2.1.2 Process algebra for biology

A seminal set of papers by Regev et al. [81, 82, 80] first proposed the use of process algebra for describing models of biochemical processes. They showed that, in the abstract, biochemical networks behaved as mobile communication systems. Molecules were treated as processes, chemical interaction treated as communication, and modification treated as channel passing. It was found that this abstraction was useful for modelling various molecular systems including transcriptional circuits, metabolic pathways, and signal transduction networks. Regev, Silverman, and Shapiro [82] showed that the π-calculus was suitable for modelling, simulating, and analysing the well-known RTK-MAPK pathway.

2.1.2.1 Quantitative models

However, Regev's approach addresses only the qualitative aspects of biochemical models. Of vital importance in the analysis of biochemical processes are the quantitative aspects: reaction times or rates, molecule counts or concentrations, etc. The next step was to extend the abstraction in the same way that the original

process algebras had been extended.

Stochastic π Priami et al. [77] showed the use of a stochastic variant of the π-calculus for modelling the quantitative properties of the RTK-MAPK pathway. The stochastic variant allows a rate of reaction to be associated with the chemical interactions, thus allowing for quantitative analysis of the system over time. It has been implemented as BioSPi[1] by the original authors and as SPiM[2] by Microsoft Research.

PEPA The interest in stochastic process calculi for modelling biological systems led to some investigations using the Performance Evaluation Process Algebra (PEPA) language of Hillston [53]. Calder et al. [22] described the use of PEPA to model the effect of RKIP on the ERK signalling pathway in two syntactically distinct, yet bisimilar, styles – the *reagent-centric* model versus the *pathway* model – demonstrating another advantage of the use of an intermediate formal language. Calder et al. [21, 20] also showed a method of deriving a system of ODEs from the PEPA models; this demonstrated the advantage of interpreting one syntactic description as more than one semantic representation. In some cases the ODE semantics are favourable and in others the stochastic semantics are favourable. It was shown, owing to Kurtz's Theorem [61] that the ODE result is a good approximation for the stochastic result.

Bio-PEPA Ciocchetta and Hillston [31] proposed an improved stochastic calculus with additional features designed for biological applications. Bio-PEPA took the *molecule as computation* abstraction a step further and takes a process to be the behaviour of a biochemical species, rather than an individual molecule. Multiple copies of a process exist in the model representing discrete levels of concentration of the species.

Bio-PEPA also makes an attempt to better handle some of the features of biological systems. Rather than treating all reactions as elementary, Bio-PEPA allows for the definition of more complex mechanisms. This is advantageous because

[1]BioSPi: `http://www.wisdom.weizmann.ac.il/~biospi/index_main.html`
[2]SPiM: `http://research.microsoft.com/en-us/projects/spim/`

reaction data from biology is often in terms of non-elementary kinetics and stoichiometry and the conversion thereof to elementary reactions is often complex. In Bio-PEPA stoichiometric coefficients can be defined for the species in a reaction and functional, parameterised rates allow the modelling of non-elementary kinetic laws such as Michaelis-Menten kinetics.

Bio-PEPA was used by Guerriero [52] to make an analysis of the Gp130 / JAK / STAT signalling pathway. Akman et al. [3] use Bio-PEPA to model a biological circadian clock. Bio-PEPA was extended by Ciocchetta and Guerriero [30] to better support the modelling of biological compartments.

Continuous π Kwiatkowski and Stark [63, 62] proposed the Continuous π-calculus (cπ) which extended the π-calculus to support *promiscuous interaction* or the ability for the reaction sites of agents to have affinity for more than one other—improving the encoding of chemical species which have reaction potential with more than one other species (the usual case). cπ also had a compositional semantics, meaning the underlying model could be composed—not just the syntactic description.

It is the Continuous π-calculus which we use for modelling biochemical processes in this thesis and it is presented in detail in Chapter 3.

Others A whole host of biological process algebras have been developed since. Regev et al. [79] extended their previous work by improving the treatment of biological compartments. In biological systems interactions are bounded by cell compartments. The expression of location in existing π-calculi was possible only by using a complex construction of private channels. To solve this problem inspiration was drawn from the Ambient calculus of Cardelli and Gordon [25] where processes are contained within *ambients*, bounding their interactions.

Designed for modelling at a cellular level, the Brane Calculus of Cardelli [24], took a more abstract view of biological compartments. The primitives of this calculus were carefully designed to reflect the biological setting and then a rich set of non-primitive operations were based on observed cellular interactions.

Priami and Quaglia [76] proposed Beta binders, an alternative biological calculus which attempts to improve on the prescribed coordination of interaction present

in previous calculi. Romanel [85] has built a dynamic biological programming environment based on a stochastic variant of Beta binders. The language, BlenX, is supported by a full toolset[3] and stochastic simulation engine. Dematté et al. [37] make a comprehensive tutorial on the language and tools. They also use these tools to drive the evolutionary study of biological pathways [38].

An alternative approach is found in rule-based languages. Danos and Laneve [35] developed the kappa-calculus (κ), a language of formal proteins, a rule-based approach to modelling protein interactions. Danos and Laneve [34] first proposed κ as a graph rewriting formalism, but the revision as an algebraic syntax improved the ability to express complexation and non-linear systems with degradation and synthesis. Koeppl et al. [60] have made a detailed study of the cyanobacterial circadian clock using κ.

The Biochemical Abstract Machine (BioCHAM) of Chabrier-Rivier et al. [28] is a rule-based language with support for compartments. The language has three separate semantic interpretations: Boolean, stochastic (population), and continuous (concentration) semantics. The Boolean semantics is the most abstract denoting only high or low presence of an object. The population semantics is a continuous time Markov chain representation allowing the tracking of individual objects. The concentration semantics is a fluid approximation of the population semantics as a system of ODEs. Cai et al. [18] used BioCHAM to model a synthetic arsenic biosensor system which addressed fatal water pollution problems. The BioCHAM toolset[4] includes comprehensive tools for biologically-oriented model checking.

2.2 Temporal logic

Temporal logic has its origins in the philosophy of language, but has been well established as a formalism for expressing temporal properties in computer science [74, 57]. Temporal logics are, in general, capable of expressing the temporal modalities: *eventually*, *always*, and *until*—and, therefore, behaviour over time. By nature of being formal languages, temporal logics have an unambiguous meaning and can therefore be used to precisely express properties in a form which is amenable to computational analysis.

[3] BlenX: `https://sites.google.com/site/aromanel/software/bwb/`
[4] BioCHAM: `https://lifeware.inria.fr/biocham/`

2.2.1 Linear Temporal Logic

A temporal logic formula (denoted by ϕ or ψ) is defined by the following grammar:

$$\phi, \psi \quad ::= \quad \mathit{Prop} \mid \phi \wedge \psi \mid \neg\phi \mid \phi \mathbf{U} \psi$$

where *Prop* is an atomic property like *server is listening* or *concentration of species A is increasing*. The formula $\phi \mathbf{U} \psi$ denotes ϕ *until* ψ, meaning that ϕ is true until the point in time when ψ becomes true. The more basic temporal modalities *eventually* and *always* can be represented by $\mathbf{F}\phi$ and $\mathbf{G}\phi$ respectively; they can be defined in terms of $\phi \mathbf{U} \psi$: $\mathbf{F}\phi \equiv \mathit{true} \mathbf{U} \phi$ and $\mathbf{G}\phi \equiv \neg\mathbf{F}\neg\phi$. Notice that $\mathbf{F}\phi$ and $\mathbf{G}\phi$ can be thought of as mnemonics for $\mathbf{F}\mathit{uture}$ and $\mathbf{G}\mathit{lobally}$ respectively and are De Morgan duals.

2.2.2 Metric/Interval Temporal Logic

Metric/interval temporal logic (MITL) [4] extends temporal logic with quantitative time information. This allows the expression of properties where precise (or bounded) timing is required, for example: *server handles request within 10ms* or *species A remains above concentration c for 5 minutes*. A review paper by Alur and Henzinger [5] survey a number of alternative approaches to temporal logic with time information. MITL has been chosen for the purposes of this thesis, the reasons for this are discussed in Chapter 4.

In MITL the temporal $\phi \mathbf{U} \psi$ becomes $\phi \mathbf{U}_I \psi$ where I is a time interval, relative to the time in its parent modality; this means that ϕ is true until the point in time when ψ becomes true, but that point in time must lie in the interval I. The time-bounded versions of the temporal modalities *eventually* and *always* can be represented by $\mathbf{F}_I\phi$ and $\mathbf{G}_I\phi$ respectively; they can be similarly defined in terms of $\phi \mathbf{U}_I \psi$: $\mathbf{F}_I\phi \equiv \mathit{true} \mathbf{U}_I \phi$ and $\mathbf{G}_I\phi \equiv \neg\mathbf{F}_I\neg\phi$. The time-bounded modalities have similar meaning to their standard counterparts, but now $\mathbf{F}_I\phi$ means *eventually* ϕ *within times* I and $\mathbf{G}_I\phi$ means *always* ϕ *within times* I.

2.3 Spatial logic

Spatial logic can refer to space at varying degrees of abstraction; for example Randall et al. [78] define a spatial logic which deal with regions and connections in topological space, Aiello and van Benthem [2] define spatial logics which deal with geometric and vector space, and Cardelli and Gordon [25] define a logic which deals with the hierarchical spatial structure of mobile computations.

It is the ideas from Cardelli and Gordon's ambient logic [25] which we will use in this thesis. Specifically, we take the *guarantee* operator—or composition adjunct—from ambient logic. The guarantee $\phi \triangleright \psi$ is satisfied only by a process for which ψ is true when it is composed with a process which satisfies ϕ. However this sort of property is difficult to compute because of the necessity to quantify over all processes which satisfy ϕ. To alleviate this and invent a computationally tractable and practical logic, in this thesis we reduce the right hand side of the guarantee to a specific process $P \triangleright \psi$—this is explained and motivated in Chapter 4.

2.4 Model checking

Model checking [33, 32], as originally defined, is an automatic technique for verifying finite-state concurrent processes. However, since then the definition has been broadened to include various kinds of processes, both finite and infinite, discrete and continuous in both time and space.

In essence, the modern definition of model checking is the computational solution to the verification, or otherwise, of the following statement:

$$P \models \phi$$

where P is a process model, ϕ is a logical formula or specification, and P satisfies ϕ.

Model checking provides a formal technique for verifying systems. A precise formal model of the system can be verified automatically and efficiently against a precise and compact specification of the desired behaviour in a formal logic.

2.4.1 Model checking in biology

A number of studies have used temporal logic for expressing temporal properties of biochemical systems and model-checking techniques to verify them. A cross-section of these include: Eker et al. [46] used temporal logic to express properties of mammalian mitogenic and stress responsive pathways; Bernot et al. [15] express properties of mucus production in *Pseudomonas aeruginosa*; Chabrier and Fages [26] apply temporal logic to gene expression regulation models; Chabrier-Rivier et al. [27] express properties of the mammalian cell-cycle control; Monteiro et al. [71] propose temporal logic patterns for analysing cell interaction networks; and Gong et al. [50] use temporal logic to verify signalling pathways in a cancer study.

All of these examples assume a discrete state space; that is the models in each of the above studies account only for molecule populations and not concentration of species. As the processes we deal with in this thesis are continuous in the nature of their state—species are measured by concentration—we need to explore some methods of model checking continuous state spaces to draw from for our own method.

2.4.2 Model checking continuous processes

Although model checking, as developed by Clarke et al., was intended to be a method of verifying models with discrete state spaces, a number of breakthroughs have been made in the area of model checking of continuous processes. The approach taken by Calzone and Fages [23, 48] is to discretize the state space and then use a logic with real-valued constraints. This approach is justified by using the discretization offered by standard numerical solvers—the rationale being that modern adaptive-step solvers give what is widely recognised to be a good approximation of the continuous solution. This approach has led to techniques for parameter estimation and fitness/robustness measures [83] and reverse engineering specifications from models [47].

Another approach, taken by Maler and Donzé [66, 44], is to use logics which are interpreted naturally over continuous domains (Boolean signals). Monitoring techniques are then used to check that a continuous signal satisfies the formula;

there are both online and offline approaches to this [67]. This approach, as well as being computationally efficient, has led to parameter estimation techniques [7] and robustness analysis techniques [40, 41].

2.5 Related work

At the time of writing, we are not aware of any other approaches to spatio-temporal logic for biochemical systems. The closest body of work is that of de Nicola and Loreti [36] who define a logic `MoMo` with a production operator which is based on the guarantee operator [25] and is similar to our context modality. However, `MoMo` is defined for mobile processes with resources and locations, modelled using a formalism based on shared tuple spaces, and thus the semantics of the production operator is quite different to the type of continuous state processes we address here.

This thesis draws heavily on much of the work we have covered earlier in this chapter. The logic we define, $\mathcal{LBC}$, is inspired by the spatial logic of Cardelli and Gordon [25]; specifically, we borrow and adapt their guarantee operator. This inspiration arose from the fact that the guarantee operator gives us both the desired means to express properties of context and the means to make use of the compositional nature of our cπ models.

The model-checking techniques we develop draw initially from the trace-based model-checking techniques of Calzone and Fages [23, 48]. We make use of their ideas on indirectly performing an approximation of model checking by using the discrete simulation trace of a model as an approximate model. We then drew inspiration for improving our own algorithm from their dynamic programming algorithm [23]. Our final hybrid algorithm drew from both our own original algorithm and their dynamic programming approach.

Our signal-based model checker for $\mathcal{LBC}$ drew from the signal-based temporal logic proposed by Maler and Nickovic [66]. The move from approximating a model by its simulation trace to approximating by a boolean signal (or a set thereof) gave a number of advantages, as discussed in Chapter 6.

Chapter 3

Continuous π-calculus

The continuous π-calculus (cπ) is a formal modelling language for the study of evolutionary variation in biochemical processes. The canonical reference for cπ is Kwiatkowski's thesis [62], but the original language semantics was first published by Kwiatkowski and Stark [63]. We have newly refined the presentation of cπ in this chapter, in order to clarify some parts of the definition and to define only the parts we need for this thesis.

The language syntax is based on the π-calculus of Milner [70], with some alterations and additions to better support the description of biochemical models. The power of π-calculus style languages for modelling biochemical processes is well-established, having first been described by Regev et al. [82, 80].

The description of a biochemical process in cπ is split into two levels: *species* and *process*. A *species* in cπ is a description of the behaviour and interaction capability of a biochemical species. This level is similar to a π-calculus process. A *process* in cπ is a real-indexed parallel composition of each of the species in the biochemical process, representing a mixture with some concentration of each species. The real index of a species in a process denotes the concentration of the species at a specific instant in time; as the process evolves over time the concentrations may change as chemical reactions occur and the process at a later instant in time may have a different set of concentration indices.

In cπ each species involved in a process is represented by a term which describes

its possible behaviours. For example, if we consider a simple chemical reaction:

$$A + B \rightarrow C + D$$

then we can define A as:

$$A \triangleq a.C.$$

That is we have a type of molecule A which has a reaction site a. It can react on a and becomes C. Then we can define B as:

$$B \triangleq b.D.$$

That is we have a type of molecule B which has a reaction site b. It reacts on b and becomes D. Below we will define how A and B interact. Now let us say that C degrades over time (possibly by some external process which we do not wish to describe, or wish to abstract away):

$$C \triangleq \tau_r.0.$$

Here C has no reaction sites. All it does is the special (autonomous) τ action at the real-valued rate r and becomes 0 (nil) which is the inert species; it has no further action. Then let us say that D is an inert species, it has no action and just accumulates in our system:

$$D \triangleq 0,$$

it behaves as the nil species.

Instead of the prescribed co-naming scheme of the π-calculus, cπ uses *affinity networks* to support arbitrary interaction. The affinity network is an undirected weighted graph of the sites with arcs recording the potential to react and the weight recording the reaction rate; thus the affinity for reaction between any biochemical species can be easily defined. Local affinity can be assigned dynamically in the case, for instance, of complexation where there is a local interaction between a bound substrate and enzyme (see example in Section 3.3).

For our simple reaction the global affinity network N will allow sites a and b to react at rate k:

$$N = \{a \overset{k}{-} b\}.$$

To complete our model we then give a process term, defining our process Π as a composition of its species and their initial concentrations:

$$\Pi \triangleq 1.0 \cdot A \parallel 1.0 \cdot B \parallel 0.0 \cdot C \parallel 0.0 \cdot D,$$

where we initially have a 1.0 Molar concentration of each of A and B, and no C or D is present.

Each cπ process evolves continuously over time through a real-valued state space. In the example with four species we have a four-dimensional space, where any point in this space represents a possible state of the system. Of course, for any given set of initial conditions only a subset of this space will be reachable. The behaviour of our system is a trajectory through this space, beginning at the point defined by the initial conditions. The trajectory for this example begins at the initial values $(1.0, 1.0, 0.0, 0.0)$, the concentrations of A and B will decrease at the same rate to zero, the concentration of C will increase initially and then decrease to zero as it degrades, and D will increase until A and B run out.

The remainder of this chapter gives the precise syntax and semantics of cπ as well as a more substantial example showing the power of cπ to express enzymatic reactions and complex formation.

3.1 Syntax

The syntax of cπ is, as we have already stated, based on that of the π-calculus [70]. However, cπ replaces co-name based communication with a symmetric and polyadic communication structure, to better represent the structure of interactions in biochemical systems. For the same reason, cπ also introduces guarded definitions and choice, and the ability to restict multiple names at once—the utility of which we will see in the example in Section 3.3.

This π-calculus-like syntax forms the *syntax of species*; this is essentially a description of the potential behaviour of a single agent in the system. On top of this is the *syntax of processes*; this is a description of the "mixture", the amount or concentration of each agent-type in the system.

The remainder of this section formally defines the syntax of species, the syntax of processes, and some basic syntactic properties.

3.1.1 Species

The syntax of species is used to define the potential behaviour of an agent. The main potential behaviour of a species is an interaction or communication; this is denoted using a communication prefix.

Definition 3.1.1 (Name)**.** A *name* (a, b, x, y, etc.) is a member of a countably infinite, totally ordered, set of names $\mathcal{N}$. The notation $\vec{a}, \vec{b}, \vec{x}, \vec{y}$, etc. is used to denote finite vectors of names.

Definition 3.1.2 (Prefix)**.** A *prefix* is a *communication prefix* of the form $a(\vec{x}; \vec{y})$ or a *silent prefix* of the form τ_k where $k \in \mathbb{R}$. The notation π, π', π_i, etc. is used to denote a prefix.

A communication prefix $a(\vec{x}; \vec{y})$ denotes potential for reaction on the channel or site a. Names in $\vec{x}$ are sent in the reaction and names in $\vec{y}$ are received. In a term $a(\vec{x}; \vec{y}).A$ names in $\vec{y}$ are binding with scope A. An occurrence of a name in a term is bound if it is, or it lies in the scope of, a binding occurrence of the name. A name is free if it is not bound (see Definition 3.1.6).

It is then necessary to define which channels or sites an agent offers have the ability to interact with which channels or sites on another agent and at what rate. This is achieved by means of an affinity network. A vertex in the affinity network denotes a channel or site and an edge denotes an interaction between sites, labelled with an associated rate.

Definition 3.1.3 (Affinity network)**.** An *affinity network* is a finite undirected weighted graph whose vertices are names and whose edges are weighted with $k \in \mathbb{R}$. An edge between two vertices a and b, labelled with k, defines an affinity between a and b with rate k. The notation M, N, K, etc. is used to denote affinity networks and the notation $M(a, b)$ denotes the rate of affinity between a and b in the affinity network M.

An affinity network can be either local or global in scope. At the top level, along with the definition of a process, we define a global affinity network which applies to the sites on all species. As we will see below, the definition of a species may also include a local affinity network, which affects only the species below it in the syntactic tree.

It is now possible to define the full syntax of a species term, thereby defining the set of species, or all possible behavioural specifications of an agent.

Definition 3.1.4 (Species)**.** The set $\mathcal{S}$ of *species* is defined by the following grammar:

$$\begin{aligned} \text{(Species) } A, B &::= 0 \mid \sum_{i=0}^{n} \pi_i.S_i \mid A|B \mid (\nu M)A \\ \pi &::= a(\vec{x};\vec{y}) \mid \tau_k \\ S &::= D(\vec{y}) \mid A \end{aligned}$$

where:

- The *nil* species 0 denotes a species which is incapable of any action.
- A definition $D(\vec{y}) \triangleq A$, where $\vec{y}$ is a vector of the free names of A, defines a species constant. The invocation $D(\vec{a})$ behaves as the body A with $\vec{a}$ substituted for the free names $\vec{y}$ of A.
- A prefix $a(\vec{x};\vec{y})$ may be denoted $a(\vec{y})$ when $|\vec{x}| = 0$, $a\langle\vec{x}\rangle$ when $|\vec{y}| = 0$, or a when $|\vec{x}| = |\vec{y}| = 0$.
- The *choice* $\sum_{i=0}^{n} \pi_i.S_i$ denotes a mutually exclusive choice of interaction; upon performing the interaction π_i the resultant species behaves as S_i. For binary choice the shorthand notation $\pi_1.S_1 + \pi_2.S_2$ may be used.
- The *species composition* $A|B$ denotes a complex of A with B.
- *Name restriction* $(\nu M)A$ restricts the scope of the names in the local affinity network M to be local in the species A. Restriction itself has no direct biochemical correspondence, but is used as a mechanism for defining local names which denote the interactions between components of a complex.

3.1.2 Process

The syntax of processes simply defines the overall system as a mixture of species, each with an initial concentration value.

Definition 3.1.5 (Process)**.** The set $\mathcal{P}$ of *processes* is defined by the following

grammar:

$$\text{(Process) } P, Q \ ::= \ c \cdot S \ \mid \ P \parallel Q$$

A *process* may be a species with an initial concentration $c \in \mathbb{R}$, or a process composition (mixture) of species.

3.1.3 Basic properties

We now define some basic properties of species and processes which will be used later in the definition of the language semantics. We define which names are free in a species, the freshness of names with respect to both species and affinity networks, and a structural congruence relation over both species and processes. We also define the notion of a prime, or non-decomposable, species.

Definition 3.1.6 (Free names)**.** The free names of a species are defined by the function $\texttt{fn} : \mathcal{S} \to \mathcal{N}$, defined recursively as:

$$\begin{aligned}
\texttt{fn}(0) &= \emptyset & \texttt{fn}(D(\vec{a})) &= \vec{a} \\
\texttt{fn}(A|B) &= \texttt{fn}(A) \cup \texttt{fn}(B) & \texttt{fn}((\nu M)A) &= \texttt{fn}(A) \backslash M \\
\texttt{fn}(\tau_k.A) &= \texttt{fn}(A) & \texttt{fn}(a(\vec{x};\vec{y}).A) &= \{a\} \cup \vec{x} \cup (\texttt{fn}(A) \backslash \vec{y}) \\
\texttt{fn}(\sum_{i=0}^{n} \pi_i.A_i) &= \bigcup_i \texttt{fn}(\pi_i.A_i)
\end{aligned}$$

Definition 3.1.7 (Freshness)**.** If X is a set of names and A is a species, then we denote by $X\#A$ that X *is fresh for* A; that is $X \cap \texttt{fn}(A) = \emptyset$. Similarly, $X\#M$ denotes that X is fresh for an affinity network M.

We define a structural congruence relation for both species and processes; this is a congruence over species and processes that, whilst syntactically different, have the same behaviour.

Definition 3.1.8 (Structural congruence)**.** A structural congruence $\equiv$ is defined over both species and processes. Structural congruence of species is the smallest congruence on $\mathcal{S}$ satisfying the following rules (left column). Structural congruence of processes is the smallest congruence on $\mathcal{P}$ satisfying the following rules

(right column).

$$
\begin{array}{rll}
0|A \equiv A & & (c \cdot 0) \parallel P \equiv P \\
A|B \equiv B|A & & P \parallel Q \equiv Q \parallel P \\
(A|B)|C \equiv A|(B|C) & & (P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R) \\
\Sigma_{i=0}^{n} \pi_i.A_i \equiv \Sigma_{i=0}^{n} \pi_{\sigma_i}.A_{\sigma_i} & \text{perm. } \sigma & (c+d) \cdot A \equiv (c \cdot A) \parallel (d \cdot A) \\
(\nu M)A \equiv A & M \# A & c \cdot (A|B) \equiv (c \cdot A) \parallel (c \cdot B) \\
(\nu M)(\nu N)A \equiv (\nu N)(\nu M)A & M \# N & c \cdot A \equiv c \cdot B \qquad A \equiv B \\
(\nu M)(A|B) \equiv A|(\nu M)B & M \# A &
\end{array}
$$

We use $\mathcal{S}^{\equiv}$ and $\mathcal{P}^{\equiv}$ to refer to the sets of $\mathcal{S}$ and $\mathcal{P}$ modulo $\equiv$, respectively.

It is also necessary to define *prime species*, that is a species which cannot be defined by a composition of non-trivial species. This is important because we need—for biochemically plausible semantics—the restriction that a single species should not model more than one independent molecule. This can be achieved by using the *prime decomposition* of a species—the multiset of prime species into which species can be decomposed.

Definition 3.1.9 (Prime species)**.** A species $A \in \mathcal{S}^{\equiv}$ is *prime* if $A \neq 0$ and if $A \equiv B|C$ then either $B \equiv 0$ or $C \equiv 0$. The set of prime species is $\mathcal{S}^{\#} \subset \mathcal{S}^{\equiv}$.

Definition 3.1.10 (Prime decomposition)**.** The prime decomposition of a species is a multiset constructed by the function $\texttt{primes} : \mathcal{S} \to \mathcal{M}(\mathcal{S}^{\#})$, defined such that:

$$\texttt{primes}(A) = \{\!|A_1, \ldots, A_n|\!\} \text{ where } A \equiv A_1|\ldots|A_n \text{ and } A_1, \ldots, A_n \in \mathcal{S}^{\#}.$$

and observing the following rules:

$$
\begin{aligned}
\texttt{primes}(0) &= \emptyset \\
\texttt{primes}(A|B) &= \texttt{primes}(A) \uplus \texttt{primes}(B) \\
\texttt{primes}(A) &= \texttt{primes}(B) \text{ when } A \equiv B.
\end{aligned}
$$

Note that, as the prime decomposition is a multiset, for $A \in \mathcal{S}^{\#}$:

$$\texttt{primes}(A|A) = \{\!|A, A|\!\}.$$

Prime decomposition is well-defined for all species and is unique up to structural congruence; the proof of which is in Kwiatkowski's thesis [63].

3.2 Semantics

The semantics of both species and processes can now be defined formally. The definition of species semantics relies on the concept of *concretions* in Section 3.2.1, then the *transition system of species* can be defined in Section 3.2.2. The *semantics of processes* is then defined in Section 3.2.3, from which a system of Ordinary Differential Equations with equivalent dynamics to the process can be computed using the technique in Section 3.2.4.

3.2.1 Concretions

To define the semantics of species, we first need to define the mechanism of concretions. Here concretions are based on Milner's concretions and abstractions [70] and combine aspects of both. A concretion can be thought of as a species which has committed to a specific interaction, but before the interaction has taken place. That is a species which can undertake an interaction becomes a concretion, if two compatible concretions exist—e.g. they are formed from two species which can interact—then they can interact to form a new species.

Definition 3.2.1 (Concretions)**.** The set of *concretions* $\mathcal{C}$ is defined by the following grammar:

$$\text{(Concretion) } F \quad ::= \quad (\vec{b};\vec{y})A \;\mid\; F|A \;\mid\; A|F \;\mid\; (\nu M)F,$$

where $A \in \mathcal{S}$, $\vec{b},\vec{y}$ are vectors of names, and M is an affinity network. The letters F and G range over concretions.

Compatible concretions interact to form species by means of *pseudo-application*. To define pseudo-application we first need to define name substitution in species.

Definition 3.2.2 (Name substitution)**.** The notation $A\{\vec{a}/\vec{x}\}$ denotes species A with all free occurrences of elements of $\vec{x}$ replaced by the corresponding elements of $\vec{a}$. It is required that $|\vec{x}| = |\vec{a}|$ and all elements of $\vec{x}$ are distinct. This is the *substitution of $\vec{x}$ by $\vec{a}$ in A*.

Definition 3.2.3 (Pseudo-application)**.** The *pseudo-application* is a binary par-

tial function $- \circ - : \mathcal{C} \times \mathcal{C} \rightharpoonup \mathcal{S}$ defined recursively as follows:

$$
\begin{aligned}
(\vec{a};\vec{x})A \circ (\vec{b};\vec{y})B &\triangleq A\{\vec{b}/\vec{x}\}|B\{\vec{a}/\vec{y}\} && |\vec{a}| = |\vec{y}|, |\vec{b}| = |\vec{x}| \\
(\vec{a};\vec{x})A \circ (F|B) &\triangleq ((\vec{a};\vec{x})A \circ F)|B \\
(\vec{a};\vec{x})A \circ (B|F) &\triangleq B|((\vec{a};\vec{x})A \circ F) \\
(\vec{a};\vec{x})A \circ (\nu M)F &\triangleq (\nu M)((\vec{a};\vec{x})A \circ F) && (\nu M)\#(\vec{a};\vec{x})A \\
(A|F) \circ G &\triangleq A|(F \circ G) \\
(F|A) \circ G &\triangleq (F \circ G)|A \\
(\nu M)(F) \circ G &\triangleq (\nu M)(F \circ G) && (\nu M)\#G
\end{aligned}
$$

The base case is undefined if $|\vec{a}| \neq |\vec{y}|$ or $|\vec{b}| \neq |\vec{x}|$, but pseudo-application is defined for any other case. When the pseudo-application $F \circ G$ is defined F and G are *compatible*, which is denoted $F \downarrow G$.

Definition 3.2.4 (Structural congruence of concretions)**.** Structural congruence $\equiv$ of concretions is the smallest congruence on $\mathcal{C}$ satisfying the following rules:

$$
\begin{aligned}
(\nu M)(A|F) &\equiv A|(\nu M)F && M\#A \\
(\nu M)(F|A) &\equiv F|(\nu M)A && M\#F \\
(\nu M)F &\equiv F && M\#F \\
(\nu M)(\nu N)F &\equiv (\nu N)(\nu M)F && M\#N \\
(\vec{b};\vec{y})(A|B) &\equiv A|(\vec{b};\vec{y})B && \vec{y}\#A \\
(\vec{b};\vec{y})A &\equiv (\vec{b};\vec{y})B && A \equiv B
\end{aligned}
\qquad
\begin{aligned}
F|0 &\equiv F \\
F|A &\equiv A|F \\
(F|A)|B &\equiv F|(A|B) \\
(A|F)|B &\equiv A|(F|B) \\
F|A &\equiv F|B && A \equiv B
\end{aligned}
$$

We use $\mathcal{C}^{\equiv}$ to refer to the set $\mathcal{C}$ modulo $\equiv$.

3.2.2 Transitions

The underlying semantics of a species is a multi-transition system, which defines all the possible actions that this species can undertake.

Definition 3.2.5 (Multi-transition system)**.** A *multi-transition system* is a tuple $(\mathcal{A}, \mathcal{L}, \mathcal{B}, \mathcal{T})$, where $\mathcal{A}$ is a set of sources, $\mathcal{L}$ is a set of labels, $\mathcal{B}$ is a set of targets, and $\mathcal{T}$ is a multiset of transitions of the form (α, λ, β), where $\alpha \in \mathcal{A}$, $\lambda \in \mathcal{L}$, and $\beta \in \mathcal{B}$. Transitions are written $\alpha \xrightarrow{\lambda} \beta$ for clarity, which denotes a transition from α to β, labelled with λ.

A multiset for transitions is required because of the quantitative nature of transitions; a transition—which is an interaction—has an associated rate. Consider the species $\tau_k.0$; it becomes 0 at rate k. Now consider the species $\tau_k.0 + \tau_k.0$; assuming standard mass action kinetics, this becomes 0 at rate $2k$. With a standard transition system the transition sets for these species would be $\{\tau_k.0 \xrightarrow{\tau_k} 0\}$ and $\{\tau_k.0 + \tau_k.0 \xrightarrow{\tau_k} 0\}$ respectively—each has the same transition. If we use a multi-transition system, where the transition multisets are $\{\!|\tau_k.0 \xrightarrow{\tau_k} 0|\!\}$ and $\{\!|\tau_k.0 + \tau_k.0 \xrightarrow{\tau_k} 0, \tau_k.0 + \tau_k.0 \xrightarrow{\tau_k} 0|\!\}$ respectively, then $\tau_k.0 + \tau_k.0$ has two transitions and we no longer lose the information that it has twice the propensity for reaction.

In the cπ multi-transition system there are three classes of transition:

Class 1 From species to concretion, labelled by a name. This represents the formation of a concretion and therefore a potential interaction.

$$A \xrightarrow{a} (\vec{b}; \vec{y})B$$

Class 2 From species to species, labelled by τ_k where k is a real number. Action occurs at rate k.

$$A \xrightarrow{\tau_k} B$$

Class 3 From one species to another, labelled by $\tau_{\langle a,b\rangle}$, where a and b are names. Action occurs at the rate of the affinity between a and b; this may (or may not) be defined in either the global or a local affinity network.

$$A \xrightarrow{\tau_{\langle a,b\rangle}} B$$

Definition 3.2.6 (cπ multi-transition system)**.** The *cπ multi-transition* system is the multi-transition system $(\mathcal{S}, \mathcal{L}, \mathcal{S} \cup \mathcal{C}, \mathit{Trans})$, where $\mathcal{L} \triangleq \mathcal{N} \cup \{\tau_k : k \in \mathbb{R}_{\geqslant 0}\} \cup \{\tau_{\langle a,b\rangle} : a, b \in \mathcal{N}\}$ and *Trans* is the set of all transitions which can be derived using the following set of rules:

$$\frac{\pi_j = a_j(\vec{b}_j; \vec{y}_j)}{\Sigma_{i=0}^{n} \pi_i.A_i \xrightarrow{a_j} (\vec{b}_j; \vec{y}_j)A_j} \text{ Choice-1}(j,n)$$

$$\frac{\pi_j = \tau_k}{\Sigma_{i=0}^{n} \pi_i.A_i \xrightarrow{\tau_k} A_j} \text{ Choice-2}(j,n)$$

$$\frac{A \xrightarrow{a} F \quad B \xrightarrow{b} G \quad F \downarrow G}{A|B \xrightarrow{\tau_{\langle a,b \rangle}} F \circ G} \text{ Com-1}$$

$$\frac{A \xrightarrow{\tau_{\langle a,b \rangle}} B \quad a,b \in M}{(\nu M)A \xrightarrow{\tau_{M(a,b)}} (\nu M)B} \text{ Com-2}$$

$$\frac{A \xrightarrow{\alpha} E}{A|B \xrightarrow{\alpha} E|B} \text{ Par-Left}$$

$$\frac{B \xrightarrow{\alpha} E}{A|B \xrightarrow{\alpha} A|E} \text{ Par-Right}$$

$$\frac{A \xrightarrow{\alpha} E \quad \alpha \notin M}{(\nu M)A \xrightarrow{\alpha} (\nu M)E} \text{ Res-1}$$

$$\frac{A \xrightarrow{\tau_{\langle a,b \rangle}} E \quad a,b \notin M}{(\nu M)A \xrightarrow{\tau_{\langle a,b \rangle}} (\nu M)E} \text{ Res-2}$$

$$\frac{B \xrightarrow{\alpha} E \quad D(\vec{y}) \triangleq B}{D(\vec{b}) \xrightarrow{\alpha\{\vec{b}/\vec{y}\}} E\{\vec{b}/\vec{y}\}} \text{ Defn}$$

where the Choice rules are rule schemes which can be instantiated for any $n \in N$ and $0 \leqslant j \leqslant n$, the letter α may denote any transition label, and E is any transition target.

3.2.3 Process semantics

The usual way of representing a continuous state space is to construct an Initial Value Problem (IVP), a set of Ordinary Differential Equations (ODEs) with some initial values. However, differential equations are not compositional; it is not possible to derive the equations for a system from the equations of its sub-systems.

For this reason the semantics of processes is defined in terms of a *process space* and a *space of potentials*, which allows the definition of system equations which are compositional. From this we can still extract, using the technique in Section 3.2.4, standard ODEs for a process.

Definition 3.2.7 (Process space)**.** *Process space* $\mathbb{P}$ is the vector space $(\mathbb{R}^{\mathcal{S}^\#}, +, \times, 0_{\mathbb{P}})$ where:

$$\mathbf{1}_A \triangleq \lambda x \in \mathcal{S}^\#. \begin{cases} 1 & A \equiv x \\ 0 & \text{otherwise} \end{cases}$$

is a unit vector in $\mathbb{P}$ and a basis is formed by the set:

$$\{\mathbf{1}_A \in \mathbb{R}^{\mathcal{S}^\#} : A \in \mathcal{S}^\#\}.$$

Definition 3.2.8 (Space of potentials)**.** The *space of potentials* $\mathbb{D}$ is the vector space $(\mathbb{R}^{(\mathcal{S}^{\#}\times\mathcal{C}^{\equiv}\times\mathcal{N})}, +, \times, 0_{\mathbb{D}})$ where:

$$\mathbf{1}_{(A,F,x)} \triangleq \lambda(B,G,y) \in \mathcal{S}^{\#} \times \mathcal{C}^{\equiv} \times \mathcal{N}. \begin{cases} 1 & A \equiv B \wedge F \equiv G \wedge x = y \\ 0 & \text{otherwise} \end{cases}$$

is a unit vector of $\mathbb{D}$ and a basis is formed by the set:

$$\{\mathbf{1}_{(A,F,x)} \in \mathbb{R}^{(\mathcal{S}^{\#}\times\mathcal{C}^{\equiv}\times\mathcal{N})} : (A,F,x) \in \mathcal{S}^{\#} \times \mathcal{C}^{\equiv} \times \mathcal{N}\}.$$

Every process in $\mathcal{P}$ can be identified with a vector in $\mathbb{P}$ which is the *phase space* of cπ models. Every dimension in $\mathbb{P}$ is a prime species and every point in $\mathbb{P}$ is a process or a state of the model. The dynamics of a cπ model is therefore a trajectory through $\mathbb{P}$. It is possible to specify a trajectory by giving the gradient vector for every point $\frac{dP}{dt}$ for every $P \in \mathcal{P}$.

However the trajectory alone—as in IVPs—is not sufficient for compositionality. The extra information is provided by the objects of $\mathbb{D}$: ∂P for every $P \in \mathcal{P}$. This is an encoding of the Class 1 (potential) transitions of the species with concentration information—a quantitative account of all the potential interactions of a process.

Definition 3.2.9 (Interaction potential)**.** The *interaction potential* of a process $P \in \mathcal{P}$ is a vector $\partial P \in \mathbb{D}$ which can be defined by structural induction on P:

$$\partial(c \cdot A) \triangleq \lambda(a,f,x). \left(c \times \mathtt{card}(a \xrightarrow{x} f, \mathit{Trans}) \times \mathtt{card}(a, \mathtt{primes}(A))\right)$$
$$\partial(P \parallel Q) \triangleq \partial P + \partial Q$$

where $\mathtt{card}(x, X)$ is the cardinality of element x in the multiset X.

In order to define $\frac{dP}{dt}$ we need a means of mapping species into process vectors and a means of combining (composing) potential objects; these are provided by the *species embedding* and the *interaction tensor* respectively.

Definition 3.2.10 (Species embedding)**.** The *species embedding* is the function $\langle - \rangle : \mathcal{S} \to \mathbb{P}$ defined by:

$$\langle A \rangle \triangleq \sum_{B \in \mathtt{primes}(A)} \mathbf{1}_B$$

Definition 3.2.11 (Interaction tensor)**.** The *interaction tensor* is a partial function $- \oplus_M - : \mathbb{D} \times \mathbb{D} \rightharpoonup \mathbb{P}$, where M is an affinity network, defined as the bilinear extension of the following clause on basis vectors:

$$\mathbf{1}_{(A,F,x)} \oplus_M \mathbf{1}_{(B,G,y)} \triangleq \begin{cases} M(x,y) \times (\langle F \circ G \rangle - \mathbf{1}_A - \mathbf{1}_B) & x, y \in M \text{ and } F \downarrow G \\ 0_{\mathbb{P}} & \text{otherwise} \end{cases}$$

The tensor takes two potentials and gives the vector in $\mathbb{P}$ which is the result of their interaction. It is now possible to give a compositional definition of $\frac{dP}{dt}$.

Definition 3.2.12 (Immediate behaviour)**.** The *immediate behaviour* of $P \in \mathcal{P}$, in the context of an affinity network M, is the vector $\frac{d_M P}{dt} \in \mathbb{P}$ which is defined by structural induction on P:

$$\frac{d_M(c \cdot A)}{dt} \triangleq \sum_{(B,k,C) \in \mathtt{taus}(A)} (k \times c \times (\langle C \rangle - \mathbf{1}_B)) + \frac{1}{2} \times (\partial(c \cdot A) \oplus_M \partial(c \cdot A))$$

$$\frac{d_M(P \parallel Q)}{dt} \triangleq \frac{d_M P}{dt} + \frac{d_M Q}{dt} + \partial P \oplus_M \partial Q$$

where $\mathtt{taus}(A) \triangleq \{(B, k, C) : B \in \mathtt{primes}(A) \wedge B \xrightarrow{\tau_k} C\}$

We can now construct a vector $\frac{dP}{dt}$ for every process P, given a global affinity network. This is equivalent to an IVP, but compositional. It is possible to extract an IVP, using the technique in the following section, in order to use standard techniques for analysis.

3.2.4 Ordinary Differential Equations

The ODEs representing a cπ system can be extracted using the following technique. It is then possible to use standard analysis techniques on the system. The technique constructs the gradient vector for each prime species in a process, as we did in the previous section, but here we abstract to *symbolic processes* by replacing real concentrations with variables. The immediate behaviour of a symbolic process is a vector of algebraic formulae, rather than a vector in $\mathbb{P}$; these algebraic formulae are the ODEs.

Definition 3.2.13 (Symbolic interaction potential)**.** The *symbolic interaction potential* $\langle \partial \rangle P$ of $P \in \mathcal{P}$ is equivalent to the interaction potential with concentration values abstracted by variables. The concentration variable c_A represents

the concentration of species A.

$$\langle\partial\rangle\,(c_A \cdot A) \triangleq \lambda(a, f, x).\left(c_A \times \mathtt{card}(a \xrightarrow{x} f, \mathit{Trans}) \times \mathtt{card}(a, \mathtt{primes}(A))\right)$$
$$\langle\partial\rangle\,(P \parallel Q) \triangleq \langle\partial\rangle\,P + \langle\partial\rangle\,Q$$

Definition 3.2.14 (Symbolic immediate behaviour)**.** The *symbolic immediate behaviour* $\left\langle \frac{d_M P}{dt} \right\rangle$ of $P \in \mathcal{P}$, in the context of an affinity network M, is equivalent to the immediate behaviour with concentration values abstracted by variables. The concentration variable c_A represents the concentration of species A.

$$\left\langle \frac{d_M(c_A \cdot A)}{dt} \right\rangle \triangleq \sum_{(B,k,C)\in\mathtt{taus}(A)} (k \times c_A \times (\langle C\rangle - \mathbf{1}_B)) + \frac{1}{2} \times (\langle\partial\rangle\,(c_A \cdot A) \oplus_M \langle\partial\rangle\,(c_A \cdot A))$$

$$\left\langle \frac{d_M(P \parallel Q)}{dt} \right\rangle \triangleq \left\langle \frac{d_M P}{dt} \right\rangle + \left\langle \frac{d_M Q}{dt} \right\rangle + \langle\partial\rangle\,P \oplus_M \langle\partial\rangle\,Q$$

The ODE extraction proceeds by first calculating the symbolic immediate behaviour. As we have noted the set of species is infinite, but a finite representation is achieved by computing the symbolic immediate behaviour for a finite set of prime species closed under transitions in *Trans*. This finite set for a process P is the set of prime species which is reachable from P.

Definition 3.2.15 (Support of a process)**.** The *support of a process* $\mathtt{supp}(P)$ of a process P is the union of the prime decompositions of each of the species in P, defined by induction on the structure of P:

$$\mathtt{supp}(c \cdot A) \triangleq \mathtt{primes}(A)$$
$$\mathtt{supp}(P \parallel Q) \triangleq \mathtt{supp}(P) \cup \mathtt{supp}(Q)$$

Definition 3.2.16 (Reachable set of prime species)**.** The set of prime species $\mathtt{reach}(P) \subseteq \mathcal{S}^{\#}$ which is reachable from a process P is defined as the least fixpoint of the function $f : \wp(\mathcal{S}^{\#}) \to \wp(\mathcal{S}^{\#})$ containing $\mathtt{supp}(P)$, where:

$$f(X) = X \cup \bigcup_{\substack{x \xrightarrow{\alpha} s \\ x \in X}} \mathtt{primes}(s) \cup \bigcup_{\substack{x \xrightarrow{\alpha} c \\ y \xrightarrow{\beta} d \\ x,y \in X}} \mathtt{primes}(c \circ d)$$

where $s \in \mathcal{S}$ and $c, d \in \mathcal{C}$.

The set $\mathtt{reach}(P)$ is not necessarily finite, even if P is syntactically finite. This is known to be true, for example, for the biochemical mechanism of polymerisation

and Kwiatkowski's thesis [62] gives an example of this. However, in most practical cases $\texttt{reach}(P)$ will be finite.

If $\texttt{reach}(P)$ is finite we can then construct a process:

$$\Pi \triangleq c_{A_1} \cdot A_1 \parallel \ldots \parallel c_{A_n} \cdot A_n$$

for each $A_i \in \texttt{reach}(P)$ and compute the symbolic immediate behaviour $\left\langle \frac{d_M \Pi}{dt} \right\rangle$. The symbolic immediate behaviour is a linear combination of the unit vectors $\mathbf{1}_{A_i}$. The ODE term for each A_i, therefore, is the sum of the coefficients of $\mathbf{1}_{A_i}$ in the term—that is, a projection on each dimension in the term.

This produces a finite set of ODEs. We can then construct an initial value problem using the initial values from P; this can be solved numerically to give a simulation trace of the dynamics of the process.

3.3 Modelling example

The following is a slightly more complicated example of modelling in cπ than our first example. It shows how enzyme-catalysed protein-protein interactions can be modelled in cπ. This concept can be used as a basis for building more complex networks of interacting proteins. It uses all of the cπ syntax and provides an intuition for the semantics of processes.

Consider the Michaelis-Menten reaction:

$$S + E \rightleftarrows ES \rightarrow E + P$$

where a substrate S binds to an enzyme E to form a complex ES. The complex can either unbind and release the substrate and enzyme, or it can react and release a product P and the unaltered enzyme. In cπ we can model this in the following way.

First we will define our simplest species, the product:

$$P \triangleq \tau_{r_{deg}}.0$$

In this case our product degrades at some rate r_{deg} and does nothing else; it becomes the inert 0 species. This is modelled as an autonomous τ reaction.

Next we model the substrate:

$$S \triangleq s(u, r).(u.S + r.P)$$

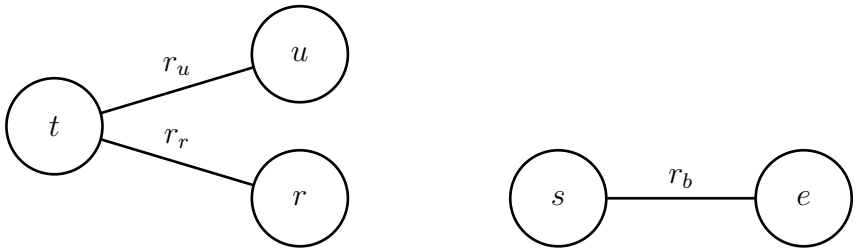
The substrate has a channel s; in cπ a channel represents a reaction site. After reacting on the site s, it has a choice of interactions: it can react on u to become the substrate again, or it can react on r to become the product. The channels u and r are received when something reacts on s.

Our final species is the enzyme:

$$E \triangleq (\nu M)\ e\langle u, r\rangle .t.E$$

The enzyme has a channel e. After reacting on e it can only do one thing: react on t and become the enzyme in its initial state again. When reacting on e the enzyme sends out two channels u and r. The channels u, r, and t are local to the enzyme; they are bound in the ν binder, which defines a local affinity graph M, shown in Figure 3.1, which allows u and t to react at rate r_u or r and t to react at rate r_r.

Figure 3.1 Local affinity network M (left) and global affinity network N (right).



Now, given we have the global affinity graph in Figure 3.1 that states that channels s and e can react, upon reaction the local channels u and r are sent on e and received on s and we form the complex:

$$ES \equiv (\nu M)\ (t.E|(u.S + r.P))$$

The complex now carries the local affinity graph from the enzyme and is a composition of the term $t.E$ from the enzyme and the term $u.S+r.P$ from the substrate. As all channels in the complex term are bound by the local affinity graph, it has no external reaction capability. The complex can only perform one of two internal reactions: t and u can react to give E and S—the unbinding—or t and r can react to give E and P—the formation of the product and release of the enzyme.

Note that the term for the complex does not need to be defined, it emerges as the result of the binding reaction.

Finally, we define the process term, which lists the species in our initial mixture and their initial concentrations $c_{i \in \mathcal{S}}$:

$$\Pi \triangleq c_S \cdot S \parallel c_E \cdot E \parallel c_P \cdot P$$

and the global affinity graph N, shown in Figure 3.1, which allows s and e to react. Note that the edges in the affinity graphs are labelled with the reaction rates: r_b for the binding rate of E and S, r_u for the unbinding rate, and r_r for the rate at which the complex forms product.

The model can now be compiled and a set of ODEs extracted using the ODE extraction technique. This results in one equation per species, both the initially defined species and any that arise from reaction—in this case, the complex:

$$\begin{aligned}
[E]' &= r_r[ES] + r_u[ES] - r_b[E][S] \\
[S]' &= r_u[ES] - r_b[E][S] \\
[P]' &= r_r[ES] - r_{deg}[P] \\
[ES]' &= r_b[E][S] - r_u[ES] - r_r[ES]
\end{aligned}$$

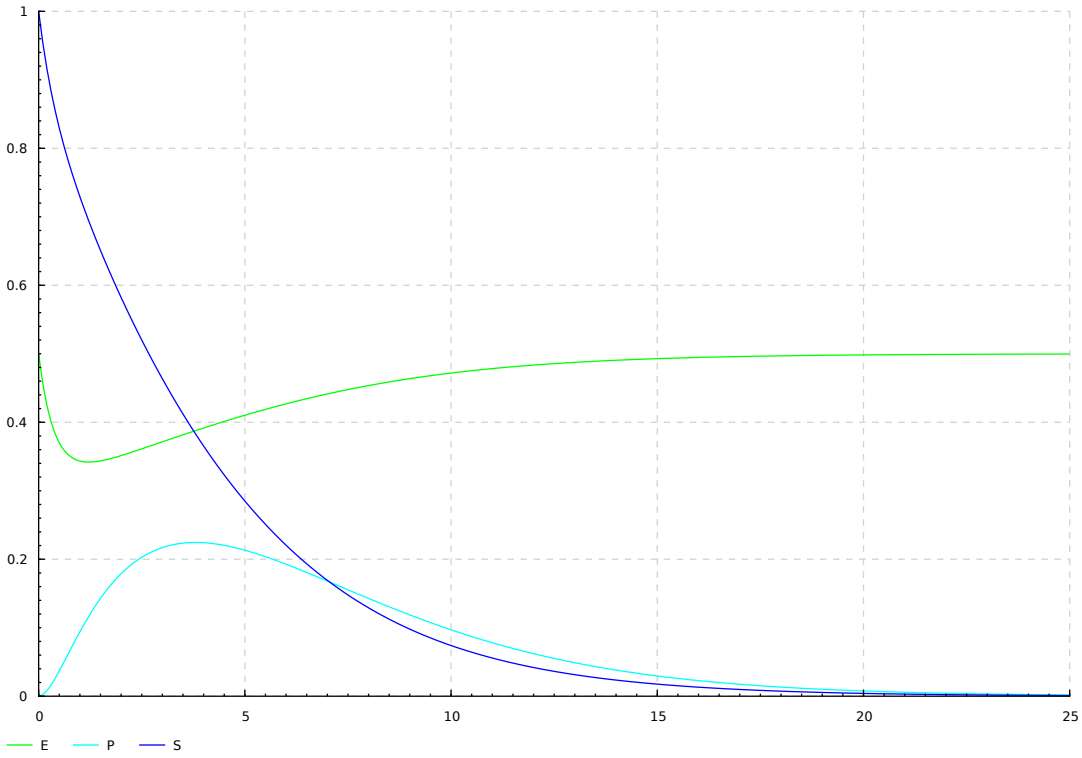
where $[A]$ is the concentration of A and $[A]'$ is the first derivative of $[A]$. The ODEs can be solved numerically, given values for the rates and initial concentrations, and we produce the time series in Figure 3.2. The values used can be found in Appendix A.

3.3.1 Larger models

The above model is essentially an abstract template for any enzyme catalysed reaction. For an example from biology, if we consider part of a MAPK signalling pathway [56], the protein kinase *Ras* promotes the phosphorylation of *Raf* to *Raf**. To model this we simply substitute *Raf* for S, *Ras* for E, and *Raf** for P in the above model; then we may substitute the rate parameters for those taken from a biological database.

It is easy to take small models of interacting proteins, like the one above, from them build larger models of interacting pathways, and from them build large

Figure 3.2 Plot of the time series of the Michaelis-Menten reaction model.



networks of interacting pathways. The compositionality of the language reduces the work needed to combine components, compared to combining ODE models which are inherently non-compositional.

We have built a number of non-trivial cπ models. For example, the MAPK signalling pathway and Kai circadian clock models built by Kwiakowski in his thesis [62]. The Kai circadian clock was also re-modelled using the more recent tools and some analysis performed in Clark et al. [11]. We have also built models of the idealised posttranslational biochemical oscillator of Jolley et al. [59] and the molecular titration models of Buchler and Louis [17]. These models demonstrate that cπ can deal efficiently with models whose behaviour is complex, e.g. oscillatory or multi-stable.

Chapter 4

Logic of Behaviour in Context

In this chapter we define the Logic of Behaviour in Context, henceforth referred to as $\mathcal{LBC}$. It is a formal language designed to precisely express properties of dynamical systems. It was designed to express properties of:

- the **state** of the agents of the system where an agent's state is a real number;
- the **temporal properties** of agents, i.e. properties of the state over (continuous) time;
- the **contextual properties** of agents, i.e. properties of the system when it is placed in the context of another system.

The motivation for defining a language which expresses both temporal and contextual properties has mainly come from biochemistry. The need to reason about the behaviour of a biochemical process when it is affected by other processes is a prime motivator for this sort of logic. However, one could conceive of many other application areas, such as reasoning about the behaviour of fluid models of distributed computer systems when they are affected by other systems.

Throughout the chapter our examples will either be abstract or they will be taken from biochemistry. For this reason, our terminology draws from both the terminology of interacting agents and from that of biochemical systems, e.g. an agent may be referred to as a species (in the biochemical sense) and the state of an agent (species) may be referred to as its concentration. We use these terminologies interchangeably, according to context.

We define $\mathcal{LBC}$ to express each of the above three types of property:

The state of an agent can be expressed by a proposition which represents a constraint on its value. For example: $[S] \leqslant c$ where $[S]$ is the value of agent S and $c \in \mathbb{R}_{\geqslant 0}$. We can then use propositional logic to express more complex constraints.

Temporal properties can be expressed using a temporal logic. A linear temporal logic is sufficient as we are dealing with deterministic systems. There is no need to quantify over paths as there is only one path in a deterministic system. However, classical linear temporal logic is not sufficient for our needs as we may wish to express properties which depend on time constraints. In particular, as we are dealing with unbounded systems, for decidability we need to bound the time frame of our properties. For this purpose we define the temporal properties of $\mathcal{LBC}$ to be similar to Metric Interval Temporal Logic (MITL) [4], where the temporal modalities are time-bounded. For example: $\mathbf{F}_{[0,t]}\phi$, where $t \in \mathbb{R}_{\geqslant 0}$, meaning eventually within t time units ϕ.

Contextual properties, which are the raisons d'être of $\mathcal{LBC}$, draw influence from spatial logics. To express contextual properties, we define the *context modality*. A context modality $Q \triangleright \phi$ holds for a model P where ϕ holds in the presence of another model Q. This is defined using a notion of model composition: P in the presence of Q is defined as $P \parallel Q$, that is P composed with Q.

The context modality is based on the guarantee operator in the spatial logic of Cardelli and Gordon's mobile ambients [25]. However, the guarantee took a logical formula on the left hand side $\phi \triangleright \psi$ meaning that the formula held for a model satisfying ψ in the presence of a model satisfying ϕ, that is:

$$P \models \phi \triangleright \psi \iff (\forall Q.Q \models \phi \implies P \parallel Q \models \psi)$$

Model checking for a logic with this guarantee would be hard—and we conjecture undecidable—for dynamical systems because of the necessity to quantify over all models that satisfy an arbitrary formula. Caires and Lozes [19] give an account of the undecidability of spatial logic with the guarantee. The context modality, however, gives some of the power of guarantee in a more tractable manner by reducing the left hand side to a specific model:

$$P \models Q \triangleright \psi \iff Q \parallel P \models \psi$$

Thus we can define a language which expresses our desired properties. This chapter continues with precise definitions of the $\mathcal{LBC}$ syntax in Section 4.1 and the semantics in Section 4.2. Then we demonstrate, by means of a series of examples in Section 4.3, the expressive power of the language. In this final section we also aim to give an intuition for the semantics of complex properties in $\mathcal{LBC}$.

4.1 Syntax

The syntax of $\mathcal{LBC}$ is that of MITL [4] with atomic propositions ranging over inequalities over the real-valued states of the agents in the system and their derivatives. To this we add the context modality $Q \triangleright \phi$.

Definition 4.1.1 ($\mathcal{LBC}$ formula)**.** The syntax of $\mathcal{LBC}$ formulae ϕ, ψ is defined by the following grammar:

$$\begin{aligned}
\phi, \psi &::= \mathit{Prop} \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \implies \psi \mid \neg\phi \\
&\quad \mid \phi \mathbf{U}_I \psi \mid \mathbf{F}_I \phi \mid \mathbf{G}_I \phi \mid M \triangleright \phi \\
\mathit{Prop} &::= \mathit{True} \mid \mathit{False} \mid \mathit{Val} \bowtie \mathit{Val} \\
\mathit{Val} &::= v \in \mathbb{R} \mid [A] \mid [A]' \mid \mathit{Val} \oplus \mathit{Val} \\
\bowtie &::= > \mid < \mid \geqslant \mid \leqslant \\
\oplus &::= + \mid - \mid \times \mid \div
\end{aligned}$$

where:

- relational operators $\bowtie$, arithmetic operators $\oplus$, and the logical operators $\wedge, \vee, \implies, \neg$ have their standard meanings;
- $[A]$ denotes the value of agent A;
- $[A]'$ denotes the first derivative of the value of agent A with respect to time;
- M is a model;
- $I \subseteq \mathbb{R}_{\geqslant 0}$ is a time interval;
- *True* and *False* denote the proposition which always holds and the proposition which never holds, respectively.

We use the abbreviations $\mathbf{U}$, $\mathbf{F}$, and $\mathbf{G}$ to denote $\mathbf{U}_{[0,\infty]}$, $\mathbf{F}_{[0,\infty]}$, and $\mathbf{G}_{[0,\infty]}$ respectively. Likewise, for $t \in \mathbb{R}_{\geqslant 0}$, we use the abbreviations $\mathbf{U}_t$, $\mathbf{F}_t$, and $\mathbf{G}_t$ to denote $\mathbf{U}_{[0,t]}$, $\mathbf{F}_{[0,t]}$, and $\mathbf{G}_{[0,t]}$ respectively.

4.2 Semantics

In this section we define the semantics of the syntactic objects we have presented in the previous section. To give a meaning to some of the objects in the syntax we must instantiate them as objects from a model. Here and throughout this dissertation we define the semantics of $\mathcal{LBC}$ formulae over cπ models. It is of course, however, possible to generalise the definition of the logic to any process model with process composition and some notion of the value of an agent.

To give a meaning to the syntactic agent A, its value $[A]$, and the syntactic model M we define in terms of cπ. An agent A is a cπ species $S \in \mathcal{S}$, whose value $[S] \in \mathbb{R}$ is its concentration, and a model M is a cπ process $P \in \mathcal{P}$ for which the composition operator $\|$ is defined.

To give a meaning to the temporal modalities we must first decide on a meaning for the time intervals attached to modalities. A metric temporal logic may have two kinds of semantics [13]: relative-time or absolute time. In a relative-time semantics the time interval of a nested temporal modality is relative to its parent. In an absolute-time semantics the time intervals of all temporal modalities are absolute with respect to the time in the model.

The two types of semantics may express different properties, with different implications for model checking. In this dissertation we will study both types. Relative-time allows the expression of properties useful for describing events and causality—as we will see in Section 4.3.1; whereas absolute-time—as we will see in Chapter 5—is more straightforward to model check.

4.2.1 Relative-time semantics

We define the relative-time semantics of $\mathcal{LBC}$ formulae by giving the satisfaction relation $\vDash$ over $\mathcal{LBC}$ formulae and cπ processes: $P \vDash \phi$ if and only if formula ϕ is satisfied by process P.

Definition 4.2.1 (atomic propositions of a c$\boldsymbol{\pi}$ process)**.** The set $Props(P)$ of atomic propositions satisfied by a process P is defined by

$$\begin{aligned} True &\in Props(P) \\ False &\notin Props(P) \\ v_1 \bowtie v_2 \in Props(P) &\iff value(v_1, P) \bowtie value(v_2, P) \end{aligned}$$

where the relational operators $\bowtie$ are defined in the normal way and

$$\begin{aligned} value(v, P) &= v \text{ for } v \in \mathbb{R} \\ value([S_i], P) &= c_i \text{ where } P = c_1 \cdot S_1 \parallel \ldots \parallel c_n \cdot S_n \\ value([S_i]', P) &= c_i' \text{ where } dP/dt = c_1' \cdot S_1 \parallel \ldots \parallel c_n' \cdot S_n \\ value(v_1 \oplus v_2, P) &= value(v_1, P) \oplus value(v_2, P) \end{aligned}$$

where arithmetic operations $\oplus$ are similarly defined as normal.

Definition 4.2.2 ($\mathcal{LBC}$ satisfaction relation)**.** For $P \in \mathcal{P}$, a cπ process, and $\mathcal{LBC}$ formulae ϕ and ψ the satisfaction relation $\vDash$ is defined inductively as follows:

$$\begin{aligned} P \vDash Prop &\iff Prop \in Props(P) \\ P \vDash \phi \wedge \psi &\iff P \vDash \phi \text{ and } P \vDash \psi \\ P \vDash \neg\phi &\iff P \nvDash \phi \\ P \vDash \phi \mathbf{U}_I \psi &\iff \text{for some } t \in I, P^t \vDash \psi \text{ and for all } t' \in [0,t], P^{t'} \vDash \phi \\ P \vDash Q \triangleright \phi &\iff (Q \parallel P) \vDash \phi \end{aligned}$$

where:

- $Q \in \mathcal{P}$ is a cπ process;
- Process P^t is the state reached from process P after time t; that is, the initial concentrations of P^t will be the component species concentrations after P has run for time t. The notation P^t is shorthand for a function mapping $\mathcal{P} \times \mathbb{R}_{\geqslant 0} \to \mathcal{P}$. Note that $P^0 = P$.

The remaining propositional connectives can be derived in the normal way and the remaining temporal modalities can be defined as follows:

Definition 4.2.3 (Derived temporal modalities)**.**

$$\begin{aligned} \mathbf{F}_I \phi &\equiv True \mathbf{U}_I \phi \\ \mathbf{G}_I \phi &\equiv \neg \mathbf{F}_I \neg \phi \end{aligned}$$

As the process term P has no time information, and nor does the result of applying P^t, at every level of recursion in the definition of $P \vDash \phi \mathbf{U}_I \psi$ the time information is lost. Therefore the time for which $\phi \mathbf{U}_I \psi$ is satisfied is relative to its parent; thus a relative-time definition. In the next section we give an absolute-time definition.

4.2.2 Absolute-time semantics

We define the absolute-time semantics of $\mathcal{LBC}$ formulae by giving the absolute satisfaction relation $\widehat{\vDash}$. A formula ϕ, interpreted in absolute-time, is satisfied by a process P if and only if $P \mathrel{\widehat{\vDash}} \phi$. We refine the relative-time semantics, with the addition of a clock $c \in \mathbb{R}$ to keep track of process time.

Definition 4.2.4 ($\mathcal{LBC}$ absolute-time satisfaction relation)**.** For a process $P \in \mathcal{P}$, a clock $c \in \mathbb{R}_{\geqslant 0}$, and formulae ϕ and ψ the absolute satisfaction relation $\widehat{\vDash}$ and a set of relations $\{\widehat{\vDash}_c \mid c \in \mathbb{R}_{\geqslant 0}\}$ are defined inductively as follows:

$$
\begin{array}{lcl}
P \mathrel{\widehat{\vDash}} \phi & \iff & P \mathrel{\widehat{\vDash}_0} \phi \\
P \mathrel{\widehat{\vDash}_c} \mathit{Prop} & \iff & \mathit{Prop} \in \mathit{Props}(P^c) \\
P \mathrel{\widehat{\vDash}_c} \phi \wedge \psi & \iff & P \mathrel{\widehat{\vDash}_c} \phi \text{ and } P \mathrel{\widehat{\vDash}_c} \psi \\
P \mathrel{\widehat{\vDash}_c} \neg\phi & \iff & P \mathrel{\widehat{\nvDash}_c} \phi \\
P \mathrel{\widehat{\vDash}_c} \phi \mathbf{U}_I \psi & \iff & \sup(I) \geqslant c \text{ and} \\
 & & \text{for some } t \in I, t \geqslant c, P \mathrel{\widehat{\vDash}_t} \psi \text{ and} \\
 & & \text{for all } t' \in [c,t], P \mathrel{\widehat{\vDash}_{t'}} \phi \\
P \mathrel{\widehat{\vDash}_c} Q \triangleright \phi & \iff & (Q \parallel P^c) \mathrel{\widehat{\vDash}_0} \phi
\end{array}
$$

where all terms in common with Definitions 4.2.1 and 4.2.2 are defined in the same way.

As we split the definition of the satisfaction relation into a set of relations, each of which relate to a clock value, the time information is carried through every level of recursion in the definition of $P \mathrel{\widehat{\vDash}} \phi \mathbf{U}_I \psi$. The absolute time information is no longer lost as in the relative-time definition. For example, the relation $P \mathrel{\widehat{\vDash}_c} \phi$ says that after running P for time c the process that it has become now satisfies ϕ, where all times in ϕ count from the start of the execution. One should also note that the context modality resets the clock.

4.3 Expressible properties

The following section is a series of examples of $\mathcal{LBC}$ which serve to give some idea of the expressive power of the language and an intuition for the semantics. The examples give an idea of the types of properties which can be expressed by $\mathcal{LBC}$. All of these first examples have the same meaning in both the relative-time and the absolute-time semantics. For examples where the chosen semantics makes a difference see the relative time examples in Section 4.3.1. We begin with some examples of the basic temporal formulae.

The basic temporal modalities are fairly easy to understand: $\mathbf{F}$ is *future* (eventually) and $\mathbf{G}$ is *globally* (always). $\mathbf{F}([A] \geqslant c)$ means eventually the value of A will be at least c. $\mathbf{G}([A] > 0)$ means the value of A will always be greater than zero, i.e. we will never run out of A.

Although $\mathbf{F}$ and $\mathbf{G}$ correspond to the classical temporal logic modalities, they can be defined in terms of the $\mathbf{U}$ modality. The $\mathbf{U}$ modality can be useful in its own right for expressing changing conditions. $([A] > 5)\mathbf{U}([B] > 1)$ means the value of A is greater than 5 *until* the value of B is greater than 1, moreover the value of B will eventually be greater than 1.

If we specify time bounds on the modalities then we can be more precise about when something is true:

- $\mathbf{F}_{24}([A] > 0)$ meaning *at some point up to time 24* the value of A will be greater than zero,
- $\mathbf{G}_{[10,15]}([A] \leqslant 1)$ meaning *between times 10 and 15* the value of A is always 1 or less.

and combining temporal modalities allows us to encode ever more complex properties, e.g.:

- $\mathbf{F}(\mathbf{G}([A] > c))$ meaning eventually the value of A will be greater than c and will remain so indefinitely.

Now we present some examples of the use of the context modality. By combining the context modality with the temporal modalities we can formulate precise statements about the behaviour of a system when we introduce something new to it. At the top level this is a fairly simple expression of a change of initial

conditions:

- $Q \triangleright \mathbf{F}([A] \geqslant c)$ meaning in the presence of Q eventually the value of A will be at least c.
- $(Q \triangleright \mathbf{F}([A] \geqslant c)) \wedge \neg \mathbf{F}([A] \geqslant c)$ is a stronger statement; not only do we state that the above property is true, but that indeed it is not true when Q is not present. Therefore we can say that it is the presence of Q which causes the value of A to become at least c.

However, if we begin to nest the context modality below the temporal modalities then we can express more complex properties about the times at which something is introduced:

- $\mathbf{G}(Q \triangleright \mathbf{F}([A] < c))$ meaning that if we introduce Q *at any point in time* then eventually $[A] < c$,
- $\mathbf{F}(Q \triangleright \mathbf{F}([A] < c))$ meaning that *there exists some point in time* such that if Q is introduced then eventually $[A] < c$,

and, again, including time bounded modalities allows one to be more precise about the time at which something is introduced:

- $\mathbf{G}_{[0,5]}(Q \triangleright \mathbf{F}([A] < c))$ meaning if we introduce Q *at any time between 0 and 5* then eventually $[A] < c$.

4.3.1 Relative time

The relative-time semantics is useful for expressing temporal properties which involve, for example events, causality, or states which occur infinitely often. Consider the following statements:

1. $\mathbf{FG}_{[0,t]}\phi$
2. $\mathbf{G}(\phi \implies \mathbf{F}_{[0,t]}\psi)$

Neither statement expresses anything particularly useful when interpreted under the absolute-time semantics. However, under the relative-time semantics, both statements express very useful properties.

Under the relative time semantics, Statement 1 means that ϕ is eventually true for at least t time units. Here the t time units is relative to the time at which ϕ becomes true. This sort of property is especially useful in a biochemical context where one might wish to state that eventually a chemical species rises to some concentration and remains so for some time.

Under the relative time semantics, Statement 2 means that whenever ϕ is true then within t time units ψ is true. This sort of statement gives us the ability to reason about events in the system and causality. If ϕ is an event, then ψ can be a—possibly delayed—response to that event. Again, this sort of property is very useful in analysing biochemical systems. Does an increase in some species cause an eventual increase in another species?

The expression of these richer relative-time properties leads on to the ability to express properties of complex dynamics; this is discussed in the next section.

4.3.2 Complex dynamics

To highlight a few more complex examples of $\mathcal{LBC}$ properties we will look at some ways to express the property of oscillation in a model. Oscillation is a difficult property to pin down in logic and we know of no wholly reliable encoding in a purely temporal logic. A good study of how to express properties of oscillation in richer than temporal logics is made by Dluhoš et al. by extending a temporal logic with a "freeze quantifier" [39].

Nevertheless it is possible to define oscillation properties in temporal logic that are less general. For example, using the relative time semantics and concentration derivatives to find regions where the concentration is increasing then decreasing and bounding these regions with intervals bounding the maximum period:

$$P \vDash \mathbf{G}_{[0,t]}(\mathbf{F}_{[0,p]}(([S]' > 0) \wedge \mathbf{F}_{[0,p]}([S]' < 0)))$$

where until time t the concentration of S is always, within time p, increasing then decreasing within time p. This captures a rising and falling of concentration, but crucially is not distinguishable from noise. It can, however, be efficiently model checked, assuming our model is known to be non-noisy. An unbounded version of this was given by Calzone et al. [23].

It is however possible using $\mathcal{LBC}$ to give a completely general expression of periodicity—to classify any periodically repeating pattern in the dynamics of a model. The key insight here is that we can use the context modality to introduce an exact copy of our model. The copy model has exactly the dynamics of the original and in no way interacts with the original. If we introduce this copy model with some time shift and the two models have coinciding dynamics then we know we have a repeating pattern with period equal to the time shift. This can be encoded thus:

$$P \models \mathbf{F}_{[p_1,p_2]}(\widehat{P} \triangleright (\mathbf{G}_{[0,t]}(|[S] - [\widehat{S}]| < \epsilon)))$$

where $\widehat{P}$ is a copy of P, S is the species being observed, and $\widehat{S}$ is the copy of S in $\widehat{P}$. This means that if we introduce $\widehat{P}$ after some period in $[p_1, p_2]$ then $[S]$ and $[\widehat{S}]$ will synchronise to within ϵ until t. Note that a flat line trivially satisfies this property, however this can easily be solved by the conjunction with the property $\neg\mathbf{G}([S]' \neq 0)$.

The beauty of this property, especially for model checking of biochemical systems, is that it is a parsimonious representation of a property which is certainly non-trivial to define in lower level computer languages. We will also see in Chapter 7 that this idea also extends to more complex dynamic properties like the identification of phase shifts due to perturbation.

4.3.3 Formal experiments in biochemistry

A good example of why a language which can express these types of properties is useful for biochemical systems can be found in any biochemical model with complex dynamics. Some good examples of models which exhibit complex, and oscillatory, behaviour are the Kai circadian clock [87] and Jolley et al.'s post-translational oscillator [59]. In Chapter 7 we use cπ and $\mathcal{LBC}$ to study these complex models. For now we will discuss some basic patterns of $\mathcal{LBC}$ which will be the foundation for our study.

Suppose we have some species S in the model, the value of which oscillates under normal conditions, and a temporal property $\texttt{Osc}([S])$ which states that $[S]$ oscillates—much like one of the properties we defined in the previous section. We

could then formulate a simple experiment by checking $R \triangleright \texttt{Osc}([S])$. This will be true if, upon the introduction of R to the model, $[S]$ still oscillates.

However, much more complex experiments could be expressed by using the context modality within a temporal modality. For example we could check, not only that $[S]$ still oscillates with the introduction of R, but that no matter at what point in the oscillation cycle we introduce R then $[S]$ still oscillates: $\mathbf{G}(R \triangleright \texttt{Osc}([S]))$. Likewise, we might state a property $\mathbf{F}_{[m,n]}(R \triangleright \neg\texttt{Osc}([S]))$ where $[m, n]$ is a specific interval of interest in the oscillation cycle. If this property is true then there exists a point in this interval where the introduction of R kills the oscillation.

One could imagine taking this idea of formulating experiments further and, for example, using some parameter fitting technique to find an interval $[m, n]$ for which our formula holds; thereby one could find a precise interval in the cycle where the introduction of R kills the oscillation.

It could be considered that the kind of discrete event simulation implemented by existing software packages such as COPASI [55] is an instance of the operational capability of the context modality. However, as the above examples should serve to show, $\mathcal{LBC}$ formulae can express much richer properties than simple discrete events by embedding these in temporal logic. Moreover, $\mathcal{LBC}$ formulae represent a succinct and formal specification of behaviour in a given context, rather than just an operation on a model.

Chapter 5

Model checking with traces

Model checking is the problem of computing whether a model M satisfies some logical specification ϕ:

$$M \models \phi$$

To check a cπ model against a specification in $\mathcal{LBC}$ we must consider the following problem: we cannot algorithmically check the whole infinite state-space. The state of the model is changing continuously over dense time. Therefore we can only reasonably compute a finite subset of the state-space. We can, however, use approximate model-checking techniques to help solve this problem.

We have developed two methods of approximate model checking for $\mathcal{LBC}$. Each method has its own means of finding a finite subset of the state-space and each method has its own advantages and disadvantages. Over Chapters 5 and 6 we describe the two techniques and analyse their relative merits. This chapter deals with the first method, using simulation traces, and Chapter 6 deals with the second method, using Boolean signals.

5.1 Trace-based model checking

Our first method for the approximate model checking of a continuous process is based on that of Calzone et al. [23] following the ideas of Antoniotti et al. [6]. First it is necessary to numerically solve the ODEs for the process model and obtain a discrete trace of the evolution of the system. A function $solve : \mathcal{P} \to Trace$ is

assumed, where a numerical ODE solver gives us a trace $t \in Trace$ of the form:

$$t = (t_0, \vec{c_0}), (t_1, \vec{c_1}), \ldots, (t_n, \vec{c_n})$$

where $t_i \in \mathbb{R}$ is a discrete time point and $\vec{c_i}$ is a vector of the concentrations of each species in the process at that time point. In this way we are checking over a finite approximation of the state-space.

For this technique of model checking over traces—and therefore each of the algorithms presented in this chapter—it is assumed that the trace is of sufficient length (in time) to allow verification of the formula. This assumption can be checked because the formula itself specifies the time interval to which it refers. Notice, though, that it is not always possible to verify a formula with an interval which is not right-closed, e.g. $\mathbf{G}_{[0,\infty)}\phi$ is not computable using this method.

Definition 5.1.1 (Duration of formula)**.** The *duration* of a formula—the length of time to which it refers —is defined recursively as follows:

$$\begin{aligned} |Atom| &= 0 \\ |\phi \wedge \psi| &= \max(|\phi|, |\psi|) \\ |\neg\phi| &= |\phi| \\ |\phi \mathbf{U}_{[a,b]} \psi| &= \max(|\phi|, |\psi|) + b \\ |Q \triangleright \phi| &= 0 \ . \end{aligned}$$

Another assumption which must be made is that the number of time points in the trace, especially where the derivative changes abruptly, is sufficient to represent an accurate approximation of the ideal model. In particular, the maximum distance between time points must be less than the minimum diameter of any interval in the temporal modalities. For example: consider a trace with time points at times $\{1, 4, 7, 10\}$ and a formula $\mathbf{F}_{[5,6]}(True)$. $\mathbf{F}(True)$ is certainly always *true*, but using the trace method we would check time point 4, find it is less than the lower bound 5, then check time point 7 and find it is greater than the upper bound 6, causing the result to incorrectly evaluate to *false*.

This assumption that we have a sufficient approximation of the state-space is harder to make. In practice, if a suitably large number of time points is requested from an adaptive step-size solver then we can have reasonable confidence in the result. However, there are no guarantees and this is a limitation of the technique.

This said, for non-critical applications it is reasonable to assume the numerical solver gives us a good approximation of the real system. Numerical ODE solvers are a well tested means of analysing dynamical systems. Indeed, the work of Calzone et al. relied on the same assumptions.

The remainder of this chapter will consider a sequence of algorithms. We begin with a simple and intuitive algorithm in Section 5.2.1 which captures the semantics, but has poor computational complexity. We then show how to improve the algorithm in Sections 5.2.2 and 5.2.3 to allow for a more tractable implementation. In each of these sections we give complexity bounds on the algorithms and in Section 5.3 we give experimental results to verify the complexity of the algorithms and the reference implementation.

5.2 Algorithms

This section details a progression of algorithms for the approximate model checking of cπ processes against specifications in $\mathcal{LBC}$, using discrete simulation traces of the cπ process. We begin with a naive algorithm which directly follows the recursive definition of the logic semantics. This algorithm is unsurprisingly inefficient, so we continue with two improved algorithms. The dynamic programming algorithm is an algorithm taken from the literature and adapted; the hybrid algorithm is a further improvement devised by combining the naive recursive algorithm with the improvements brought by the dynamic programming algorithm.

These algorithms are all limited to the absolute time semantics of $\mathcal{LBC}$ which was defined in Section 4.2.2. Initially this was for simplicity, and indeed the naive algorithm can be extended to treat the relative time semantics. However, the dynamic programming technique can only deal with absolute time, so in this chapter we treat only the absolute time semantics. In Chapter 6 we go on to define an efficient algorithm for the relative time semantics.

5.2.1 Naive

A minimal, naive algorithm for model checking $\mathcal{LBC}$ over traces is defined quite succinctly by the functional pseudocode in Figure 5.1. We follow directly the

Figure 5.1 Functional pseudocode for naive model-checking algorithm.

```
check :: Trace → Formula → Bool
check (t:ts) (atom) = valid atom t
check ts (φ ∧ ψ)
   = (check ts φ) AND (check ts ψ)
check ts (¬φ)
   = NOT (check ts φ)
check (t:ts) (φ U_[t0,tn] ψ)
   = case time(t) of
     < t0 : check ts φ U_[t0,tn] ψ
     ⩽ tn : check (t:ts) ψ
            OR (check (t:ts) φ
                AND check ts φ U_[t0,tn] ψ)
     otherwise : False
check (t:ts) (Q ▷ φ)
   = check (solve (compose Q (proc t))) φ
```

recursive definition of the $\mathcal{LBC}$ satisfaction relation.

The base case for atomic propositions calls a function `valid` which simply checks that the constraints on species concentrations are satisfied at the current time point in the trace. For example, with $[A] \geqslant 0.1$ it checks the value of c_A corresponding to time point t is greater than or equal to 0.1.

Checking a context modality involves taking the current process (`proc`), composing it with the introduced process (`compose`), computing the trace for this new process (`solve`), and checking the introduction's sub-formula over this new trace.

The worst-case time complexity for this naive algorithm is exponential in the size of formula. That is $O(n^f)$ where n is the length of trace and f is the depth of nested temporal ($\mathbf{U}$) formulae. This is owing to how the recursion unfolds for nested $\mathbf{U}$ formulae. This is true for just the temporal fragment of the logic and therefore for checking the full logic $\mathcal{LBC}$.

If we consider $\mathbf{G}(\mathbf{F}\phi)$ where ϕ does not become true until the end of the trace then we see that the trace is traversed n times, $\mathbf{F}\phi$ being re-checked unnecessarily at each time point. In the next section we present an algorithm which eliminates this unnecessarily repeated computation.

Figure 5.2 Dynamic programming model-checking algorithm.

1. By post-order depth-first traversal of the formula we obtain sub-formulae ordered by dependency.
2. Reverse the ordering of the trace.
3. We traverse the reversed trace once, labelling each time-point with each sub-formula in order if it holds, according to the following rules:
   - An atomic proposition holds if its constraint is satisfied.
   - $\phi \wedge \psi$ holds if the time-point is already labelled with both ϕ and ψ.
   - $\neg\phi$ holds if the time-point is not already labelled with ϕ.
   - $\phi \mathbf{U}_{[t_0,t_n]} \psi$ holds if:
     - *either* the time is $< t_0$ and the previous time point is labelled with $\phi \mathbf{U}_{[t_0,t_n]} \psi$.
     - *or* the time is $\leqslant t_n$ and:
       - *either* the time-point is already labelled with ψ
       - *or* it is already labelled with ϕ and the previous time point was labelled with $\phi \mathbf{U}_{[t_0,t_n]} \psi$.
   - $Q \triangleright \phi$ holds if the following procedure returns *true*:
     (a) construct the cπ process Π of this time point,
     (b) solve $Q \parallel \Pi$ and apply the algorithm to this new trace and ϕ.
4. If the initial time-point is labelled with the whole formula then return *true*, otherwise return *false*.

Note that as well as the problem of exponential complexity, in practice the cost that is incurred by solving the differential equations is also highly significant. We will discuss this further in Section 5.4.

5.2.2 Dynamic programming

The dynamic programming algorithm used by Calzone et al. [23] is an established method for model checking temporal logic over traces and a partial answer to this problem. By traversing the trace only once, checking each sub-formula of the formula at each time point, it avoids unnecessary re-traversals of the trace.

We extend the algorithm to treat the context modality, which proceeds as in Figure 5.2.

Note that in step 1 the context modality is treated as an atomic formula. The sub-formula of a context modality is only used by the new call of the algorithm in computing its satisfaction for the trace of the newly composed process.

Now the trace is only traversed once if we are checking the fragment of $\mathcal{LBC}$ without the context modality. Therefore, the worst case for this fragment is polynomial in the size of formula $O(nf)$, a significant improvement.

If we consider the full logic, then we still see exponential worst-case complexity $O(n^f)$, because the new process created by a context modality necessitates re-computation. However, improvement is seen in some cases over the naive algorithm. In the naive algorithm we see exponential complexity in the depth of **U** nesting. In the dynamic programming algorithm we eliminate the re-computation of directly nested temporal modalities, hence polynomial time for the temporal fragment.

One immediate disadvantage of this algorithm is that, although we only traverse the trace once, we always traverse the whole trace. The recursive algorithm, working forwards along the trace, will terminate if it finds, say, a witness for $\mathbf{F}\phi$ or a counterexample to a $\mathbf{G}\phi$ before the end of the trace. Indeed, to check an atomic proposition only the initial time point is required. This *short-circuiting* can save a lot of computation in practice.

Worse, the dynamic programming algorithm's lack of short-circuiting means that when checking a context modality the re-computation of a trace is necessarily done at every time point. This is true even, for example, when the context modality is at the top level.

With this in mind, in the next section we describe a further improved, hybrid, algorithm.

5.2.3 Hybrid

By combining ideas from the previous two algorithms we can devise a hybrid algorithm. We take the recursive algorithm, add the aspect of dynamic programming in a recursive manner, whilst retaining the ability to short-circuit the checking. We also make use of memoisation for optimisation.

The algorithm is based on the naive algorithm, but instead of evaluating over the structure of formulae we evaluate over a list (or array) of sub-formulae. Rather, we have an outer recursion over the trace and an inner recursion over the list of sub-formulae. As in the dynamic programming algorithm, this list should be in dependency order, a post-order depth-first traversal of the formula.

To avoid re-evaluating sub-formulae that we have already evaluated we can memoise the result of evaluating each sub-formula at each time point, i.e. we build a lookup table mapping $Formula \to \mathbb{B}$ for each time point. As we evaluate sub-formulae in dependency order, we ensure that any previously computed value is always in the lookup table. This is essentially what the dynamic programming approach does by labelling time points with their satisfied formulae.

The algorithm is presented as functional pseudocode in Figure 5.3. The `check` function now takes a list of the sub-formulae of the formula to be checked, a trace, and returns a lookup table (`Table`) of sub-formulae and whether they are satisfied. If the whole formula is in the returned lookup table, with a value of *true*, then it is satisfied for the trace.

The function `check` ranges over the time points and the function `checkSt` ranges over the sub-formulae, using the function `eval` to determine the satisfaction of a sub-formula at a given time point. A value is retrieved from the lookup table by `lookup` and inserted by `insert`. Note that the lookup as presented in Figure 5.3 should always find a result in the table—as we are evaluating in post-order a sub-formula will always be evaluated before its parent. We also have a function `subs` which gives a list of the sub-formulae of a formula. The functions `valid`, `solve`, `compose`, and `proc` are the same as in Section 5.2.1.

The `check` and `checkSt` functions must use a lazy, or call-by-need, evaluation strategy to implement the short-circuiting. The `check` function calls `checkSt` to build the lookup table for each sub-formula at time point t; this call includes the result of calling `check` for the remaining time points $(t{+}1, \ldots, t{+}n)$. Under a call-by-need evaluation strategy, if the result for future time points is not needed then it will not be evaluated—effectively implementing the short-circuiting. Likewise the `checkSt` function inserts the evaluation of a sub-formula into the lookup table containing the evaluations of the next sub-formulae in the list; a call-by-need strategy means that later computations may not need to be evaluated.

Figure 5.3 Functional pseudocode for hybrid model-checking algorithm.

```
check :: [Formula] → Trace → Table
check fs []     = emptyTable
check fs (t:ts) = checkSt t (check fs ts) fs

checkSt :: State → Table → [Formula] → Table
checkSt t next []     = emptyTable
checkSt t next (f:fs) = insert f (eval fs t now next f) now
                        where now = (checkSt t next fs)

eval :: State → Table → Table → Formula → Bool
eval t now next (atom)
   = valid atom t
eval t now next (φ ∧ ψ)
   = (lookup φ now)
     AND (lookup ψ now)
eval t now next (¬φ)
   = NOT (lookup φ now)
eval t now next (φ U_[t0,tn] ψ)
   = case time(t) of
     < t0 : lookup (φ U_[t0,tn] ψ) next
     ⩽ tn : (lookup ψ now)
                OR ((lookup φ now)
                     AND  (lookup (φ U_[t0,tn] ψ) next))
     otherwise : False
eval t now next (Q ▷ φ)
   = lookup (Q ▷ φ) (check (subs φ) (solve (compose Q (proc t))))
```

For the temporal fragment we still have the same worst-case complexity as the dynamic programming algorithm, but we reduce the likelihood of hitting the upper bound by exploiting short-circuiting. We evaluate over no more of the trace than is needed to return a result, e.g. if we find that ϕ is true at the start of a trace and we are evaluating $\mathbf{F}\phi$ then we stop as soon as we find this witness, without evaluating over the rest of trace. This algorithm also has the advantage that we are evaluating forwards along the trace and there is no need to reverse the trace.

However, we do not eliminate the re-computation required where nested temporal modalities are separated by a context modality. Therefore, for the full logic the hybrid algorithm can do no better than $O(n^d)$ where d is the *sandwich alternation depth*, which we define as the greatest nesting of temporal modalities separated by context modalities. This is close to, but not the same as, classic alternation depth. For example: $Q \triangleright \mathbf{G}\phi$ only requires a single trace traversal $O(n)$; so does $\mathbf{G}(Q \triangleright \phi)$ and even $Q \triangleright \mathbf{G}(Q' \triangleright \phi)$; but $\mathbf{G}(Q \triangleright \mathbf{G}\phi)$ requires multiple trace traversals $O(n^2)$ and $\mathbf{G}(Q' \triangleright \mathbf{G}(Q \triangleright \mathbf{G}\phi))$ requires $O(n^3)$.

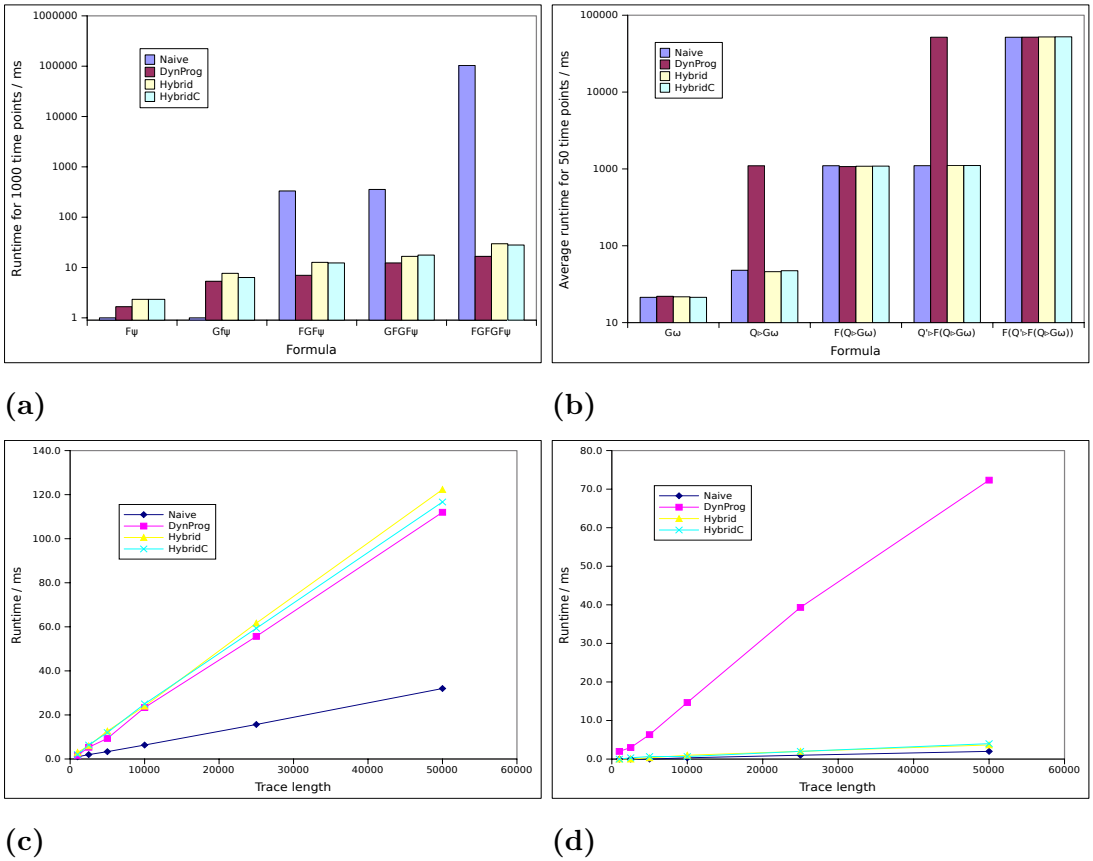
5.3 Complexity and profiling

We have implemented cπ, $\mathcal{LBC}$, and the various model-checking algorithms thereof. For more information on the implementation see Chapter 8. Using this reference implementation, some experimental results were produced which support the assertions which we have made about complexity in the preceding section. We show that the experimental results match the expected complexity bounds, therefore verifying—in the informal sense—both the complexity and the efficiency of the implementation.

Throughout this section we make use of basic formulae of the following form. ϕ is a proposition which is true towards the start of the trace, ψ is a proposition which is true only towards the end of the trace, χ is a proposition which is only false towards the end of the trace, and ω is false towards the start of the trace.

In the next two section we summarise the performance profiling results and compare them to the expected complexity. Full details of the model used in the experiments, along with parameters and propositional formula values, appear in

Figure 5.4 Trace-based model checker performance results: (a) runtimes for increasing temporal formula depth, (b) runtimes for increasing sandwich alternation depths of $\mathcal{LBC}$ formulae, (c) runtimes for a formula with no short-circuit potential, and (d) runtimes for a formula with short-circuit potential. In (a) and (b): ϕ is true towards the start of the trace, ψ is true only towards the end of the trace.



Appendix A. Full systematic profiling results appear in Appendix B.

5.3.1 Temporal logic fragment

First we examine the performance results of model checking just the temporal logic fragment of $\mathcal{LBC}$. The naive algorithm has a runtime which is exponential in the alternation depth of the temporal modalities; Figure 5.4a illustrates this. Formula depth is the greatest depth of alternating temporal modalities in the formula: e.g. $\mathbf{F}\phi$ has depth 1, $\mathbf{GF}\phi$ has depth 2, etc.; $\mathbf{G}\phi \wedge \mathbf{F}\phi$ has depth 1.

The performance plots show runtimes for implementations of the naive algorithm,

the dynamic programming algorithm, and two implementations of the hybrid algorithm. The second implementation of the hybrid algorithm (HybridC) uses a lazy circular data structure in Haskell to implicitly implement the memoisation; it can be seen that this gives roughly the same performance as the more explicit implementation.

The dynamic programming algorithm does not allow the short-circuiting described in Section 5.2.2. Figure 5.4d shows runtimes for checking a formula $\mathbf{F}\phi$ where ϕ is a proposition which is true towards the start of the trace. Clearly the naive algorithm, in its simplicity, has the best performance in this case and the dynamic programming and hybrid algorithms have comparable, worse performance. Figure 5.4c shows runtimes for the formula $\mathbf{F}\psi$ where ψ is a proposition which is true only towards the end of the trace. Now, the naive and hybrid algorithms perform much better than the dynamic programming algorithm which does not short-circuit.

The full results of a systematic exploration of runtimes for varying formulae are shown in Figure B.1 in Appendix B.

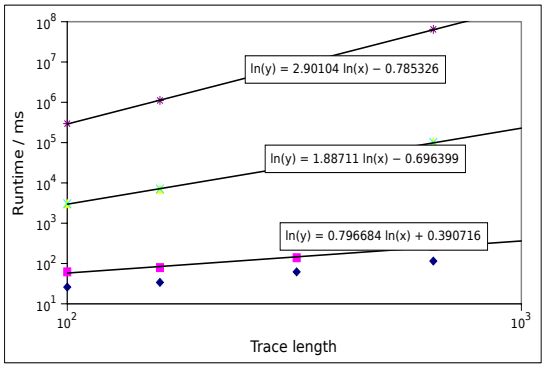
5.3.2 $\mathcal{LBC}$

Now we examine the performance results for model checking all of $\mathcal{LBC}$, including the context modality. Figure B.2 in Appendix B shows that, for most cases, the dynamic programming algorithm has a much greater runtime; its lack of short-circuiting is critical when we are forced to minimise the considerable cost of calling the ODE solver.

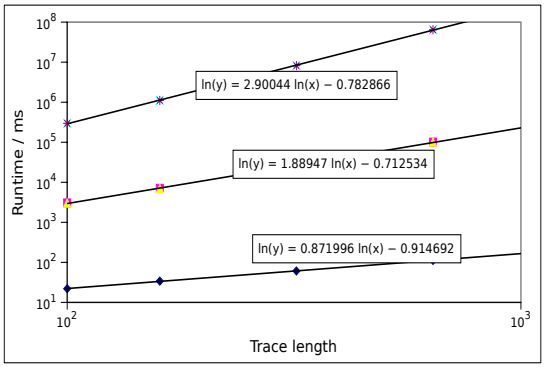
Figure B.2 also shows that the naive algorithm has no worse runtime than the hybrid algorithm for most cases when we include the context modality. However, when we combine the context modality and nested temporal modalities, the naive algorithm has the same exponential increase in computation time incurred by nesting temporal modalities.

In Section 5.2.3 we noted that complexity increases exponentially in the *sandwich alternation depth*; Figure 5.4b illustrates this. The dynamic programming algorithm increases ahead of the recursive algorithms, again, because of the lack of short-circuiting.

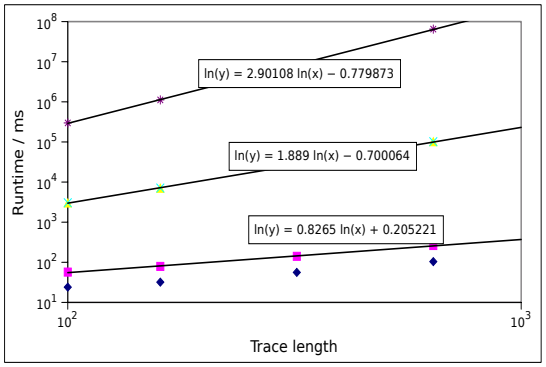
Figure 5.5 Runtime exponent increase with sandwich alternation depth.



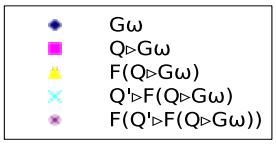
(a) Naive



(b) Dynamic programming



(c) Hybrid



(d) Legend for (a),(b),(c).

Figure 5.5 shows that the runtime exponent does indeed correspond to the sandwich alternation depth for the hybrid algorithm. From the plots we can see that the runtimes increase with approximately the predicted exponents. For example, in Figure 5.5c, running the hybrid algorithm on a formula $\mathbf{F}(Q' \triangleright \mathbf{F}(Q \triangleright \mathbf{G}\omega))$ with sandwich alternation depth 3 for increasing trace length fits the function $\ln(y) = 2.90108 \ln(x) - 0.779873$ which has a gradient of approximately 3. For all plots in Figure 5.5c the correlation coefficient R^2 is greater than 0.99. We can also see that the dynamic programming algorithm increases its exponent with the addition of a context modality, rather than a context modality nested within temporal modalities.

5.4 Limitations

A limitation of the trace-based technique is the assumption that the number of time points in the trace, especially where the derivative changes abruptly, is

sufficient to represent an accurate approximation of the ideal model. It is possible to miss an accurate approximation of the precise point in time by having too few time points in the trace. A partial solution to this is the event detection technique discussed in Section 10.3, which finds, to some known accuracy, the switching points of the basic (atomic proposition) signals. However, when checking the context modality $P \models Q \triangleright \phi$ with this technique we are only checking $P^t \parallel Q \models \phi$ for t in the original trace. We can only assume that, once again, the trace is dense enough to guarantee this for all t. A potential solution to this problem is discussed in Chapter 9.

Also, in the previous section, we saw improvements in the computation time of the model-checking algorithm. However, this does disregard a major constant factor in practice. The calls made to the ODE solver by checking a context modality are computationally heavy, especially with complex models.

The key to improving the efficiency of model checking is reducing the number of calls we have to make to the solver. We can go some way to reducing the number of calls to the solver by re-writing the formula before model checking and eliminating context modalities. For example:

$$(Q \triangleright \phi) \wedge (Q \triangleright \psi) \mapsto Q \triangleright (\phi \wedge \psi)$$
$$Q \triangleright (Q' \triangleright \phi) \mapsto (Q \parallel Q') \triangleright \phi$$

The formula on the right-hand side of each rule is logically equivalent to the formula on the left, but with fewer nested context modalities. Each of these rules, when applied, will reduce the number of calls to the solver, required to model check the formula, from two to one.

We also see multiple solver calls when increasing the sandwich alternation depth and formula rewrites cannot eliminate any of the calls required by sandwich alternation. Consider, for example, $\mathbf{G}(Q \triangleright \phi)$ where for every time point in the original trace we need to introduce Q and call the solver to obtain a new trace for the system composed with Q, that is $n+1$ calls to the solver. These new calls to the solver, however, are independent and therefore can be executed concurrently. Parallelising the algorithm should give speedup linear with the number of threads. However, an improved sequential technique is discussed in Chapter 9.

Chapter 6

Model checking with signals

In the previous chapter we explored a series of algorithms for the approximate model checking of $\mathcal{LBC}$ using simulation traces. Those algorithms were limited to the absolute time semantics of $\mathcal{LBC}$; the reason for this was twofold, to initially treat a simple case and to make use of an existing algorithm to help solve the problem. In this chapter we show how the relative time semantics can be checked.

We have seen, in Section 4.3.1, how a relative time semantics for $\mathcal{LBC}$ is necessary for expressing a large class of useful properties. Relative time properties are especially useful in our target application area of biochemical models. In this chapter, therefore, we devise a method for efficiently checking such properties.

An established technique for giving semantics to metric interval temporal logics—which generally have relative time—is to use Boolean signals as a model. The technique was pioneered by Maler and Nickovic's definition of Signal Temporal Logic [66]. In Section 6.1 we use this technique by converting simulation traces into Boolean signals and define a signal checking procedure which accounts for the context modality.

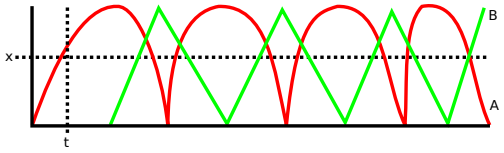
6.1 Signal-$\mathcal{LBC}$

The basic idea behind signal-based approximate model checking is that the dynamics of the model is represented as a set of Boolean signals; each basic signal represents whether an atomic proposition in a formula is satisfied at a given time.

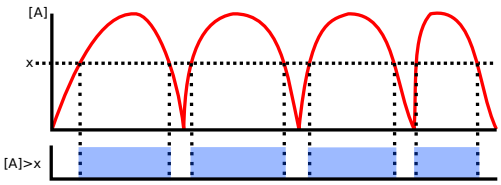
By defining combinators over these basic signals, and having these combinators related to the logical operators, we can find the satisfaction of non-atomic formulae.

We will define these concepts formally in the following sections, but first we give an intuitive example. Consider the simulation trace for the concentrations of two chemical species A and B in Figure 6.1a and the formula we wish to check $\mathbf{F}_{[0,t]}((A > x) \vee (B > x))$—within time t either A or B will be greater than x. The basic signals in Figures 6.1b and 6.1c represent the times at which each atomic proposition, $[A] > x$ and $[B] > x$, is true.

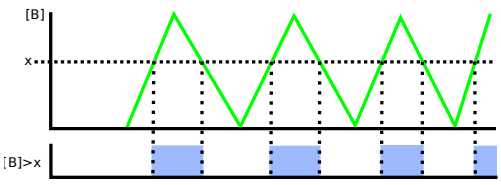
Figure 6.1 A simulation trace and its splitting into Boolean signals.



(a) Simulation trace.



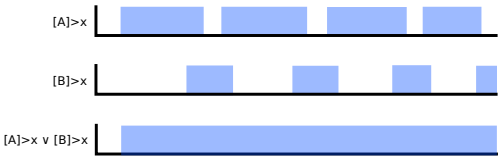
(b) Signal for $[A] > x$.



(c) Signal for $[B] > x$.

The signal for the whole formula can be built up using signal combinators as follows. We first compute the signal for $(A > x) \vee (B > x)$ by applying the signal combinator for $\vee$, the union of the signals, as shown in Figure 6.2a. The signal for the whole formula is then computed by the combinator for $\mathbf{F}_{[a,b]}$, a back-shifting of the start of each positive interval in the signal by b and the end by a (ignoring negative values), as shown in Figure 6.2b. We can conclude that the formula is satisfied because its signal is true initially.

Figure 6.2 Boolean signal combinators applied.



(a) Signal combination for $\vee$.

(b) Signal combination for $\mathbf{F}_{[0,t]}$.

However, our problem is how to define the Boolean signal for the context modal-

ity $P \models Q \triangleright \phi$. The approach we take here is to take a set of sample points within the time horizon of the formula, testing $P^t \parallel Q \models \phi$ for each sample point t, and compressing this down to a signal. This approach, however, still suffers from the limitation discussed in Section 5.4 and a potential solution to this is discussed in Chapter 9.

6.1.1 Signals

Signals are constructed from the dynamics of a model. A signal represents the satisfaction of a formula at any given time. The set *Signal* is the set of finite length Boolean signals.

Definition 6.1.1 (Finite length Boolean signal)**.** A *finite length Boolean signal* s is a function $s : [0, r) \to B$ where $r \in \mathbb{R}_{\geqslant 0}$ and s switches between *True* and *False* only finitely many times over its range.

This finite variability is equivalent to giving s a finite *interval covering*: a sequence $\mathcal{I} = I_1, I_2, \ldots$ of left-closed right-open intervals such that $\bigcup I_i = [0, r)$ and $I_i \cap I_j = \emptyset$ for all $i \neq j$. The covering $\mathcal{I}_s$ of the signal s is *consistent* with the signal if $s(t) = s(t')$ for all t, t' in the same interval $I_i \in \mathcal{I}_s$. A covering $\mathcal{I}'$ is a *refinement* of $\mathcal{I}$, denoted $\mathcal{I}' \prec \mathcal{I}$, if for all $I' \in \mathcal{I}'$ there exists $I \in \mathcal{I}$ such that $I' \subseteq I$. The set of *positive* intervals of s is $\mathcal{I}_s^+ = \{I \in \mathcal{I}_s : s(I) = \mathit{True}\}$ and the set of *negative* intervals is $\mathcal{I}_s^- = \mathcal{I}_s \setminus \mathcal{I}_s^+$.

Signal checking relies on the conversion from the dynamics of the model to a set of basic signals. The basic signals represent the satisfaction of the atomic propositions of a formula.

6.1.1.1 Basic signals

Basic signals are constructed from a simulation trace, from the set *Trace* of simulation traces of the form:

$$(t_0, \vec{c_0}), \ldots, (t_n, \vec{c_n})$$

where t_i is a time point and $\vec{c_i}$ is a vector of the species concentrations at that time. To construct the basic signals we use the following procedure:

1. Take each leaf ϕ in the syntax tree of the formula; these are the atomic propositions of the form $[A] \bowtie c$, $[A]' \bowtie c$, *True*, or *False*.
2. For each ϕ we construct a signal s_ϕ as an interval covering $\mathcal{I}$ of intervals $[t_0, t_1), [t_1, t_2), \ldots, [t_{n-1}, t_n)$.
3. Each interval $[t_i, t_{i+1})$ is in $\mathcal{I}^+_{s_\phi}$ if the constraint in ϕ is satisfied by the values in $\vec{c_i}$, otherwise it is in $\mathcal{I}^-_{s_\phi}$.

This set of signals gives the satisfaction of the atomic propositions of a formula over time.

6.1.1.2 Signal combinators

For non-atomic formulae there is a set of signal combinators which take the basic signals, apply a logical operation, and give the signal for the satisfaction of a formula over time. A signal s is constructed by computing its covering intervals $\mathcal{I}_s$; it is sufficient to compute the positive intervals $\mathcal{I}^+_s$ as the negative intervals $\mathcal{I}^-_s$ are, by definition, complementary.

Definition 6.1.2 (Boolean signal combinators [66])**.** The signal combinators apply the logical connectives $(\neg, \wedge)$ and temporal modalities $(\mathbf{F}, \mathbf{U})$ to signals (s_ϕ, s_ψ) and are defined as follows:

$\neg s_\phi$

Negation is a simple negation of the signal such that $\mathcal{I}^+_{s_{\neg\phi}} = \mathcal{I}^-_{s_\phi}$.

$s_\phi \wedge s_\psi$

For conjunction we first compute a refinement of the coverings $\mathcal{I}^R_\phi \prec \mathcal{I}_\phi$ and $\mathcal{I}^R_\psi \prec \mathcal{I}_\psi$ such that $\mathcal{I}^R_\phi = \mathcal{I}^R_\psi$ and is the sequence of intervals $I^R_1, \ldots, I^R_n$. The conjunction is then computed interval-wise such that $s_{\phi\wedge\psi} = s_\phi \wedge s_\psi$. The minimal covering $\mathcal{I}_{s_{\phi\wedge\psi}}$ is then computed by merging any adjacent intervals of the same Boolean value.

$\mathbf{F}_{[a,b]} s_\phi$

The temporal $\mathbf{F}_{[a,b]}$ modality is computed by back-shifting the positive intervals. $\mathcal{I}^+_{\mathbf{F}_{[a,b]}\phi}$ is constructed by taking each interval $I \in \mathcal{I}^+_\phi$ and computing its back-shifting $I \ominus [a, b] \cap \mathbb{R}_{\geqslant 0}$ where $[m, n) \ominus [a, b] = [m - b, n - a)$ and the intersection with $\mathbb{R}_{\geqslant 0}$ eliminates any negative times. The minimal cover-

ing $\mathcal{I}_{\mathbf{F}_{[a,b]}\phi}$ is then computed by merging any adjacent intervals of the same Boolean value.

$s_\phi \mathbf{U}_{[a,b]} s_\psi$

The fundamental temporal $\mathbf{U}_{[a,b]}$ modality can be computed on the basis that $\phi \mathbf{U}_{[a,b]} \psi \iff \phi \wedge \mathbf{F}_{[a,b]}(\phi \wedge \psi)$ when s_ϕ is a unitary signal. A signal s is unitary if $\mathcal{I}_s^+$ is a singleton. So if s_ϕ is unitary and it holds at t_1 and t_2 then it must hold for the whole interval $[t_1, t_2]$. For the case where s_ϕ is not unitary we can decompose it into a set of unitary signals $\{s_\phi^1, \ldots, s_\phi^n\}$ and compute, for each $i \in [1, n]$:

$$s^i_{\phi \mathbf{U}_{[a,b]} \psi} = s^i_\phi \wedge \mathbf{F}_{[a,b]}(s^i_\phi \wedge s_\psi)$$

The signal is then recomposed to give:

$$s_{\phi \mathbf{U}_{[a,b]} \psi} = \bigvee_{i=1}^{n} s^i_{\phi \mathbf{U}_{[a,b]} \psi}$$

6.1.1.3 Context modality signal

Our problem is how to compute the signal for the context modality $Q \triangleright \phi$. The positive intervals of the signal $\mathcal{I}^+_{s_{Q \triangleright \phi}}$ must represent the times at which if Q is composed with the model then ϕ is satisfied. To compute this we must choose a finite number of arbitrary time points at which to introduce Q to the model and compute the satisfaction of ϕ. The problem lies in how to choose these time points.

An initial solution is to choose the same time points as in the original trace; that is if we are checking $P \models Q \triangleright \phi$ then we use the same time points as in the trace for P. The assumption here is that if the chosen time points for P were sufficiently dense then they will be sufficiently dense for $Q \triangleright \phi$.

Definition 6.1.3 (Context modality signal)**.** The context modality signal is constructed as follows. To compute a signal for $P \models Q \triangleright \phi$, for each time point t in the originally computed simulation trace we compute a new process $P^t \parallel Q$. Each of these new processes is solved numerically to get a trace and we recursively apply the signal checking procedure to find whether or not ϕ holds for each of these processes. The Boolean result from each process at time t is the value of the signal for the interval $[t, t')$ where t' is the time of the next point.

6.2 Complexity and Profiling

This section is an analysis of profiling the model checker implementation using signals, along with relevant comparison with the implementation based on traces. For the purpose of comparison we use the same model and formulae as in Section 5.3. The difference in semantics can be ignored here as we concentrate on the raw, systematic performance results for varying complexity of formula and trace. The trace checking algorithm used for comparison is the best performing hybrid algorithm.

Throughout this section we make use of basic formulae of the following form. ϕ is a proposition which is true towards the start of the trace, ψ is a proposition which is true only towards the end of the trace, χ is a proposition which is only false towards the end of the trace, and ω is false towards the start of the trace.

Full details of the model used in the experiments, along with parameters and propositional formula values, appear in Appendix A.

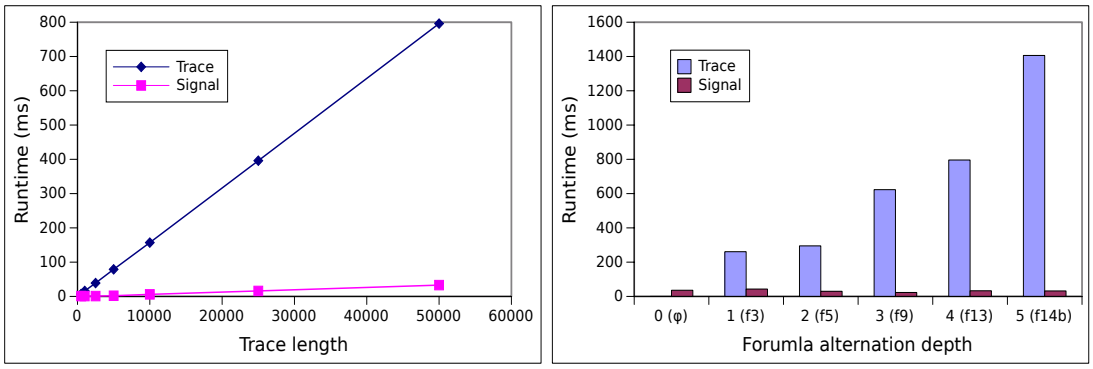
6.2.1 Temporal logic fragment

For the temporal fragment of $\mathcal{LBC}$ the performance of the signal-based checker is a clear improvement over the trace-based checker. The trace checker gave an increase in runtime which was linear in the trace length. Whilst the signal checker also exhibits a linear relationship, it increases with a greatly reduced constant factor; Figure 6.3a shows this relationship.

For the trace checker, increasing the alternation depth of the formula increased the runtime linearly. However, for the signal checker the performance analysis shows that the runtime with increasing formula alternation depth increases with a smaller constant factor then the trace checker; Figure 6.3b shows this relationship.

The improvement in performance over both trace length and formula size can be ascribed to the compression achieved by converting a trace into a signal. Once the signal has been calculated, the runtime depends on the density of the signal; density depends not on the trace length, but on the number of atomic propositions in the formula and the number of times the truth value of those propositions changes of the length of the signal. In practice, the signal density is usually a

Figure 6.3 Comparison of runtime performance for trace and signal checking over (a) increasing trace length and (b) increasing formula size, for the temporal fragment of $\mathcal{LBC}$.



(a) **(b)**

very small number of intervals in comparison the number of time steps in the parent trace.

The full results of a systematic exploration of runtimes for varying formulae are shown in Figure B.3 in Appendix B.

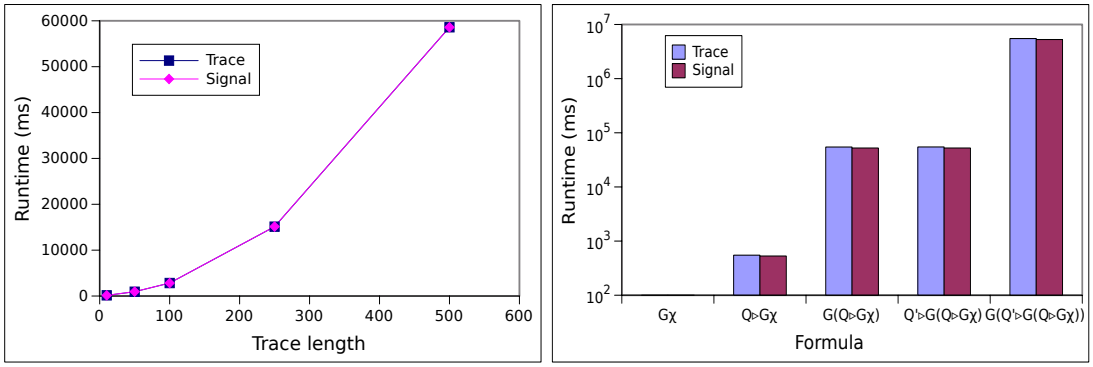
6.2.2 $\mathcal{LBC}$

The performance of the signal checker for the full $\mathcal{LBC}$ is comparable to the trace checker. Figure 6.4a shoes the performance with respect to trace length, in the worst case, is approximately equivalent for the trace checker and for the signal checker.

The signal checker, like the trace checker, has runtime exponential in the sandwich alternation depth of the formula. Figure 6.4b shows results for the initial implementation, which show runtime increasing in line with the trace-based checker. In the worst case for both algorithms signal checking does have the same complexity as the trace checking for $\mathcal{LBC}$.

The full results of a systematic exploration of runtimes for varying formulae are shown in Figure B.4 in Appendix B. The full results highlight that whilst the signal checker does short-circuiting on temporal formulae, this is not possible for the context modality signal because the whole signal needs to be computed. Because the whole signal needs to be computed—so it can possibly be combined

Figure 6.4 Comparison of runtime performance for trace and signal checking over (a) increasing trace length and (b) increasing sandwich alternation depth, for the full $\mathcal{LBC}$.



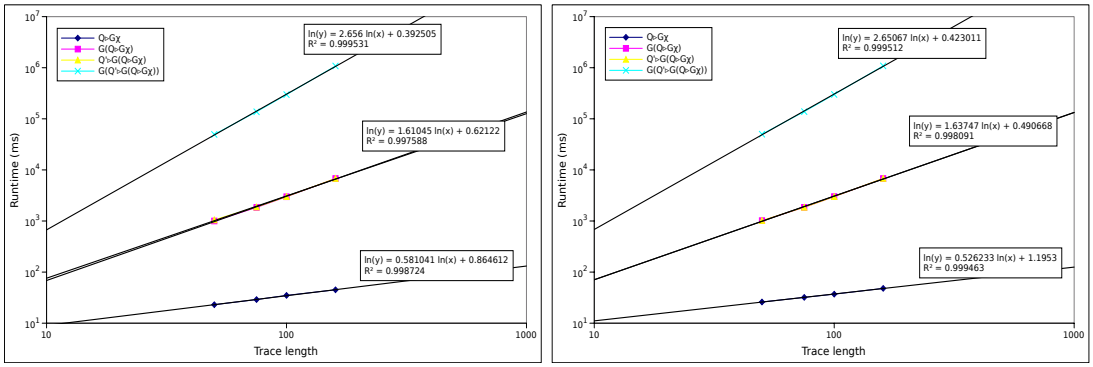
(a) **(b)**

with another signal—the solver needs to be called for every time point. The ideas using sensitivity analysis in Chapter 9 attempt to provide a means to address this problem.

6.2.3 Complexity estimation

Figures 6.5a and 6.5b show the estimation of the runtime exponent increase with sandwich alternation depth, the same worst case as the trace checker.

Figure 6.5 Runtime exponent increase with sandwich alternation depth for (a) the trace checker and (b) the signal checker.



(a) **(b)**

6.3 Conclusions

Performance of the signal checker is significantly better for the temporal fragment of the logic in almost all cases. For the full logic $\mathcal{LBC}$ the worst case performance is comparable to the performance of the trace checker, but it does not share the trace checker's short-circuiting property. This is a small penalty to pay for the more practical expressiveness of the relative time semantics.

The runtime increases with sandwich alternation depth to the same degree as the trace checker. These results are especially encouraging given that the relative time semantics of the signal checker allows expression of more properties which are useful in application. However, we can still do no better than sandwich alternation for model checking the full logic and we lose short-circuiting for context modality signals. Both of these problems are addressed to some degree in Chapter 9 where we will introduce a technique which has the potential to help reduce the number of calls to the solver in practice.

Chapter 7

Post-translational oscillator case study

In this chapter we describe the encoding of a post-translational oscillator (PTO) model in cπ and the results of computational experiments made on the model. This includes the use of $\mathcal{LBC}$ to specify and check properties of the model.

A PTO is a mechanism of the circadian clock in organisms, a system in which species concentrations vary regularly with a twenty-four hour period. PTOs form part of the larger system in an organism that regulates the day-night cycle of other systems. Although there are other kinds of circadian oscillator, post-translational oscillators are based solely upon the modification of proteins—often phosphorylation—and not by gene regulation.

The model under investigation is the theoretical PTO of Jolley et al. [59]. The structure of the model is relatively simple, but capable of producing complex behaviour. It is designed to show that a very simple structure—just one molecule with two phosphorylation sites—can exhibit robust oscillatory behaviour similar to that of known, real circadian clocks.

The purpose of our study is to further examine the behavioural properties of the PTO when it is coupled with other PTOs, other reaction pathways, and inhibitors. We examine these properties using both simple computational experiments—in Section 7.3—and more complicated, *higher-order* experiments defined by $\mathcal{LBC}$ properties and performed by model checking—in Section 7.4. The ultimate goal being to evaluate $\mathcal{LBC}$ as a useful logical tool to perform these sorts of analyses

and also to draw some conclusions about the behaviour of the theoretical PTO in relation to real circadian oscillators—in Section 7.5. In particular circadian clock mechanisms must interact with other systems in an organism; this includes the control of metabolic processes and coupling with the classical transcription-translation feedback loop (TTFL) circadian clocks [1]. This potential to robustly interact with other systems is, to date, unexplored for the theoretical model.

One of the main advantages of using a process algebra, such as cπ, is that the model description is compositional; this allows the modeller to easily alter the interaction structure and makes the process of coupling two models almost trivial. This is much more difficult when using a non-compositional description like ODEs. Another advantage is that the algebraic syntax of the model description has an unambiguous interpretation with respect to the structure of biochemical interactions and more closely relates to interaction structure than an ODE description.

The advantage of using $\mathcal{LBC}$ to specify and check properties of the model, and its composition with other models, is that it gives us a concise and precise means of expressing the hypothesis we wish to test. This is especially true where we have a mixture of temporal and spatial behaviour we wish to test; e.g. if we wish to know if an inhibitor introduced at any point in an oscillation cycle always has some effect.

7.1 The Jolley model

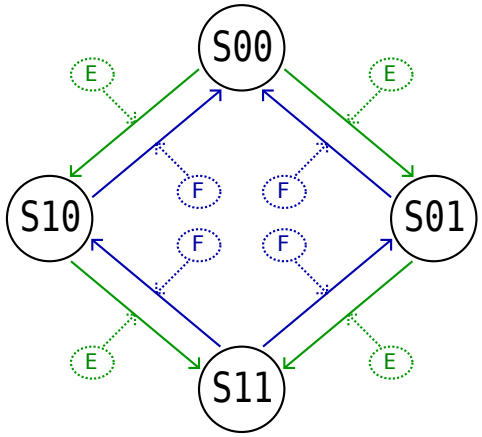
Jolley, Ode, and Ueda present their model as a set of coupled ODEs. In their paper [59], two sets of parameters are identified which give two distinct patterns of oscillation in the system. The model aims to provide a framework for analysing and synthesising PTOs and they provide evidence that it is a viable candidate for a minimal circadian clock. However, to date, little analysis of the properties of the complex behaviour of this oscillator has been done.

The model arose from the observation that PTOs and other oscillatory systems which exist in nature are commonly mediated by multi-site phosphorylation, these include evidence from observations and existing models of the KaiC circadian oscillator [72, 58, 87], the MAP Kinase signalling pathway [29, 65], and others [59].

This motivated the search for the simplest possible phosphorylation-mediated oscillator, to serve as a design principle.

The structure of Jolley's PTO (jPTO), described diagrammatically in Figure 7.1, is one molecule with two phosphorylation sites. Therefore the molecule has four states (S00, S01, S10, S11) depending on which of its sites are phosphorylated. Two opposing enzymes, a kinase (E) and a phosphatase (F), act to phosphorylate or dephosphorylate a site, respectively.

Figure 7.1 Structure of jPTO, showing the four substrate molecule states, the kinase E, and the phosphatase F.



The parameters for this model were found by using computational parameter fitting techniques. They then used a clustering algorithm to determine two distinct clusters of parameter sets which produced two different patterns of oscillation. The clusters which were found share the motif that complete cycles between the states are restricted to one direction; Figure 7.2 illustrates this. Figure 7.3 shows the behaviour of the model given representative parameter sets from each cluster.

Figure 7.2 Illustrating the relative reaction rates for each jPTO cluster. A thicker arrow denotes a faster reaction. A dotted arrow denotes a reaction so slow as to be effectively disabled.

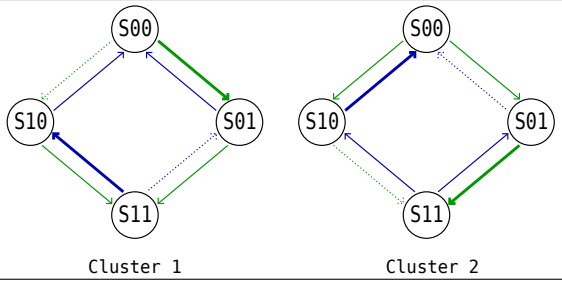
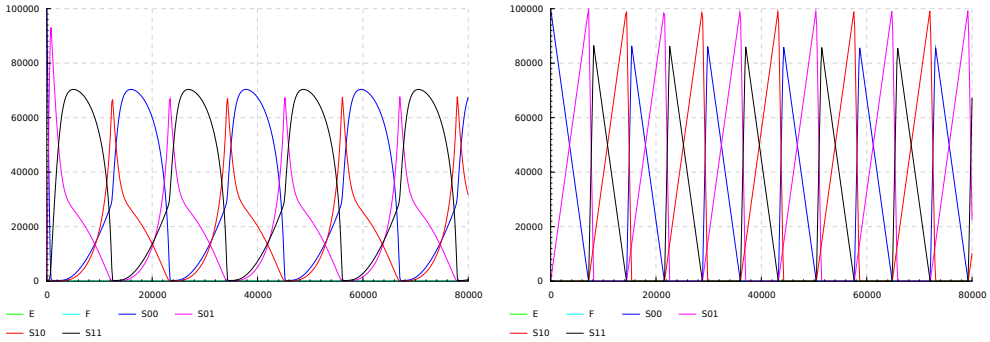


Figure 7.3 Oscillatory dynamics of jPTO.



(a) Parameter cluster 1. **(b)** Parameter cluster 2.

Jolley et al. perform a number of analyses on the model. The analysis includes a study of temperature compensation which shows the jPTO to be robust to changes in temperature, showing that the model is a viable candidate circadian oscillator.

7.2 cπ model construction

Model construction in cπ is species-centric. That is the biochemical species, or reagents, are the focus of the modelling process. We first define each species and its binding sites and actions. We then define how different species can interact with each other. Then we define the initial conditions of our mixture, which species are present and in what concentrations. The model can then be executed to determine the behaviour, using numerical simulation. The remainder of this section gives an overview of the construction and execution of the model in cπ.

7.2.1 Species

The species in our model are the kinase E, the phosphatase F, and the substrate molecule which has four phosphorylation states $S00$, $S01$, $S10$, and $S11$. The simplest of these are the two enzymes; they are defined as follows:

$$E \triangleq e(x).x.E$$
$$F \triangleq f(x).x.F$$

The kinase E has a site e and the phosphatase F has a site f. Each can interact on its site with another molecule, perform some other function which depends on the molecule it is bound to, then return to its original state—from which it can perform the same action again. This directly corresponds to the definition of an enzyme.

In our cπ model we represent each of the four states of the substrate as a distinct species. This is simply to break down the syntactic description into smaller parts. In this model a change of state is essentially a change of species, but to the observer these species can be considered as one. The substrate can be defined as follows:

$$\begin{aligned}
S00 &\triangleq (\nu M_{00})\ s00a\langle be\rangle.(u.S00 + ra.S01) \\
&\qquad + s00b\langle be\rangle.(u.S00 + rb.S10) \\
S01 &\triangleq (\nu M_{01})\ s01e\langle be\rangle.(u.S01 + r.S11) \\
&\qquad + s01f\langle bf\rangle.(u.S01 + r.S00) \\
S10 &\triangleq (\nu M_{10})\ s10e\langle be\rangle.(u.S10 + r.S11) \\
&\qquad + s10f\langle bf\rangle.(u.S10 + r.S00) \\
S11 &\triangleq (\nu M_{11})\ s11a\langle bf\rangle.(u.S11 + ra.S01) \\
&\qquad + s11b\langle bf\rangle.(u.S11 + rb.S10)
\end{aligned}$$

Here each of the states is defined, each containing a definition of the behaviour at each of the two phosphorylation sites. Each of these definitions is similar in structure, reflecting that they in fact represent distinct states of the same molecule. For example, let us examine the definition of $S01$.

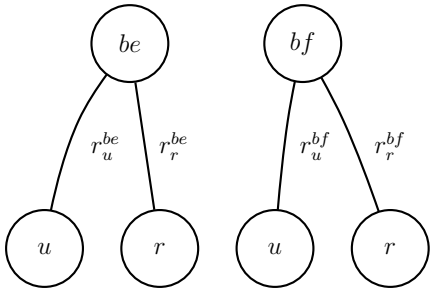
One of the two states where one site is phosphorylated, but not the other, is $S01$. The term begins with a ν-term. The ν-term defines a local affinity network M_{01}; this governs the local interactions of unbinding or reacting in the same way as the global affinity network which will be defined below. M_{01} is the graph shown in Figure 7.4 where each arc has an associated rate of reaction; this defines the internal interaction potential of the complexes formed between substrate and enzyme to unbind (u) or react (r).

The structure of $S01$ is then defined as having two sites $s01e$ and $s01f$, each with some behaviour which follows from another molecule binding on that site. Once we have defined which molecules can interact on which sites (below), $s01e$ will

accept the kinase E and $s01f$ will accept the phosphatase F. The behaviour which follows binding is defined by the next part of the term; in this case the bound enzyme can either unbind and the substrate returns to state $S01$ or the reaction can occur, changing the substrate either to state $S00$ or to $S11$, depending on whether F or E is bound.

The definition of each of the other states of the substrate follow the same pattern. Full details of definitions, the affinity graphs, and their rates can be found in Appendix C.

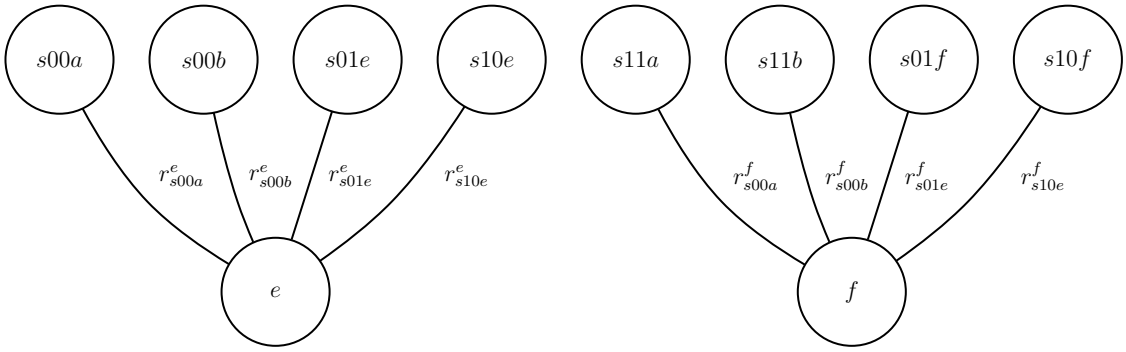
Figure 7.4 Local affinity graph M_{01}.



7.2.2 Interactions

Now we have the definitions of the molecules and their interaction sites, we need to define which molecules can bind to which sites and at what rate these reactions occur. This is done by means of an affinity network M, shown in Figure 7.5. Here we state that each of the substrate sites interacts with either site e of the kinase or site f of the phosphatase. Each of these interactions has a given reaction rate (see Appendix C).

Figure 7.5 Global affinity graph M.



7.2.3 Mixture

Having now defined the structure and rate parameters of the model, all that remains to be able to execute the model is a definition of the initial conditions we wish to simulate. Here we define a process Π which lists the species present and their initial concentrations.

$$\Pi \triangleq c_S \cdot S00 \parallel c_E \cdot E \parallel c_F \cdot F$$

Here we have some concentration c_S of substrate in its unphosphorylated state $S00$ and likewise some concentrations c_E and c_F of E and F.

7.2.4 Validation

Upon execution, the cπ interpreter generates the underlying model semantics. From this a set of ODEs and an initial value problem can be derived and numerically simulated to give the behaviour of the model. In this case the model description generates precisely the set of ODEs which were defined by Jolley et al. and therefore precisely the same behaviour; as shown in Figure 7.3.

7.3 Basic time series analysis

In this section we describe a number of computational experiments which were performed, aided by the compositional nature of the cπ description of the model. A number of these experiments were performed to illuminate the complex behavioural properties of the jPTO.

This section, whilst it seems at first not immediately relevant for this thesis, provides the reader with an overview of the complex dynamics of the system and its interactions. It also serves to show how far we can explore with the standard analysis and to allow the reader to better see the benefits and added power of using $\mathcal{LBC}$ for analysis. Moreover, it is the case that analysis using $\mathcal{LBC}$ should not be considered to be a stand-alone technique, but part of a wider analysis toolset. We also aim to provide better intuition for the properties which we will come to express in $\mathcal{LBC}$.

First, in Section 7.3.1, we examine the behaviour of coupling jPTOs. How does one affect the other? How does the way in which jPTOs are coupled affect the behaviour? These questions are of interest primarily because in real circadian clocks it has been observed to be the case that robust clocks are constructed by the coupling of one or more less robust clocks [1, 86].

Next, in Section 7.3.4, and for the same reason, we examine the effect of coupling jPTOs when their oscillations are not in phase. This sheds further light on the robustness of the clock. Do the jPTOs synchronise when they are coupled out of phase? Or do they dampen each other?

We then examine how the jPTO can interact with other reaction networks. In Section 7.3.5 we allow the jPTO to catalyse the phosphorylation of another type of molecule. In a real circadian system the clock would drive a number of other reaction pathways. How does driving another reaction affect the clock?

Finally, in Section 7.3.6, we examine how exposing the jPTO to an inhibitor for a short time affects the system. Is it robust to perturbation? This illuminates another type of robustness in the system. Examining how a short perturbation affects the system at varying points in its phase cycle gives phase response characteristics of the jPTO. Phase response [51] provides useful information on oscillators and their synchronisation properties.

7.3.1 Coupled jPTOs

The first experiment determines the behaviour of two identical jPTOs when coupled. The coupling is achieved by the two jPTOs sharing a pool of enzymes E and F.

To achieve the coupling in our cπ model in the following way. First we make a copy of the substrate species, renaming the states Sxx to Txx and sites $sxxx$ to

$txxx$:

$$\begin{aligned}
T00 &\triangleq (\nu M_{00})\ t00a\langle be\rangle.(u.T00 + ra.T01) \\
&\qquad + t00b\langle be\rangle.(u.T00 + rb.T10) \\
T01 &\triangleq (\nu M_{01})\ t01e\langle be\rangle.(u.T01 + r.T11) \\
&\qquad + t01f\langle bf\rangle.(u.T01 + r.T00) \\
T10 &\triangleq (\nu M_{10})\ t10e\langle be\rangle.(u.T10 + r.T11) \\
&\qquad + t10f\langle bf\rangle.(u.T10 + r.T00) \\
T11 &\triangleq (\nu M_{11})\ t11a\langle bf\rangle.(u.T11 + ra.T01) \\
&\qquad + t11b\langle bf\rangle.(u.T11 + rb.T10)
\end{aligned}$$

The process term can then be updated to include our new T jPTO:

$$\Pi \triangleq c_S \cdot S00 \parallel c_T \cdot T00 \parallel c_E \cdot E \parallel c_F \cdot F$$

and the global affinity network M can then allow E and F to interact with the sites of T. Here we use a (more compact) textual representation of the affinity network, full details are in Appendix C.

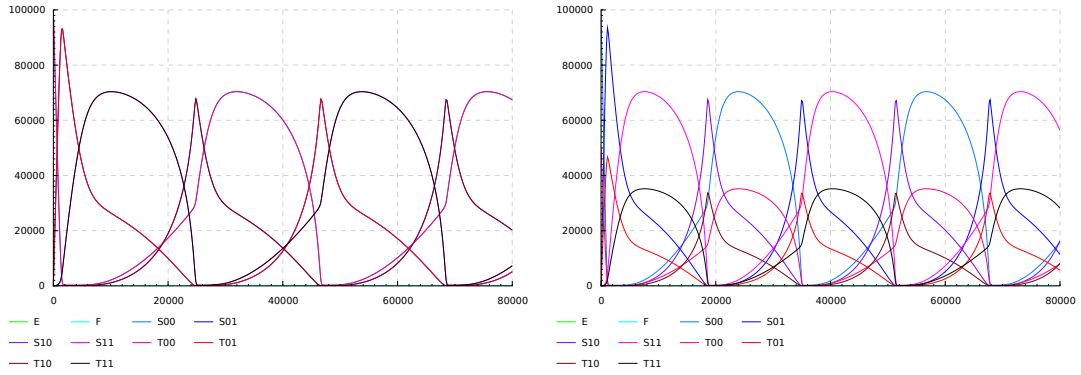
$$\begin{aligned}
M = \{&s00a \leftrightarrow e,\ \ s00b \leftrightarrow e,\ \ s01e \leftrightarrow e,\ \ s10e \leftrightarrow e, \\
&s01f \leftrightarrow f,\ \ s10f \leftrightarrow f,\ \ s11a \leftrightarrow f,\ \ s11b \leftrightarrow f \\
&t00a \leftrightarrow e,\ \ t00b \leftrightarrow e,\ \ t01e \leftrightarrow e,\ \ t10e \leftrightarrow e, \\
&t01f \leftrightarrow f,\ \ t10f \leftrightarrow f,\ \ t11a \leftrightarrow f,\ \ t11b \leftrightarrow f\}
\end{aligned}$$

Each clock individually behaves as in Figure 7.3a. The behaviour of the coupled jPTOs can be seen in Figure 7.6a.

The result of coupling two identical jPTOs is that the two act in synchrony, but the period is doubled. It is clear that the doubling of the period is due to each jPTO only having half the concentration of enzymes available, the other half of the concentration being sequestered by the other jPTO—each is competing equally over the same pool.

If we take a jPTO with half the substrate concentration (call it jPTO/2) then the period of oscillation is halved. If we then couple jPTO and jPTO/2, again sharing the enzyme pool, we see that we still achieve synchronisation; see Figure 7.6b. The resultant period is proportional to the normal periods of the two jPTOs.

Figure 7.6 jPTO composition sharing an enzyme pool.



(a) Coupling two identical jPTOs. **(b)** Coupled jPTO and jPTO/2.

7.3.2 Weaker coupling

It is possible to consider other schemes for coupling. For example, if the coupling was made weaker by only sharing one of the enzymes, does synchronisation still occur?

Here we take two jPTOs in a similar manner to above, however we only share the kinase E. This is achieved in the model simply by having a separate phosphatase for each jPTO, F_S and F_T:

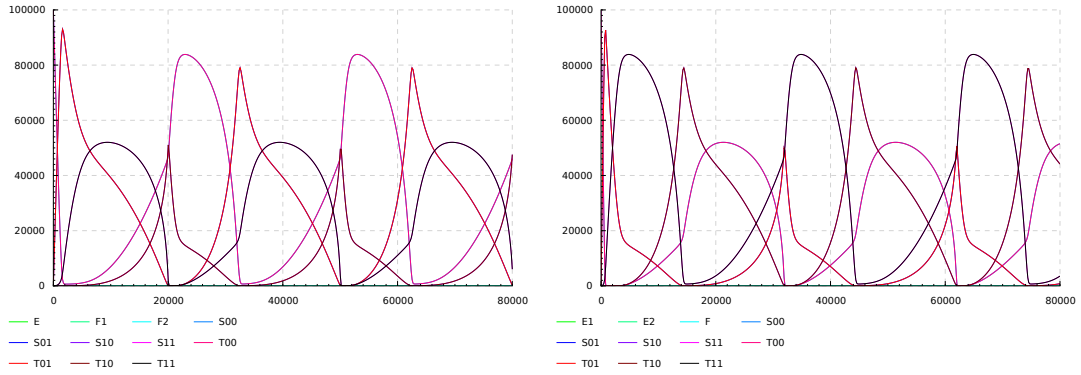
$$\Pi \triangleq c_S \cdot S00 \parallel c_T \cdot T00 \parallel c_E \cdot E \parallel c_{F_S} \cdot F_S \parallel c_{F_T} \cdot F_T$$

Here we set $c_{F_S} = c_{F_T} = c_E$. We then set the global affinity network accordingly:

$$\begin{aligned} M = \{ & s00a \leftrightarrow e,\ \ s00b \leftrightarrow e,\ \ s01e \leftrightarrow e,\ \ s10e \leftrightarrow e, \\ & s01f \leftrightarrow fs,\ \ s10f \leftrightarrow fs,\ \ s11a \leftrightarrow fs,\ \ s11b \leftrightarrow fs \\ & t00a \leftrightarrow e,\ \ t00b \leftrightarrow e,\ \ t01e \leftrightarrow e,\ \ t10e \leftrightarrow e, \\ & t01f \leftrightarrow ft,\ \ t10f \leftrightarrow ft,\ \ t11a \leftrightarrow ft,\ \ t11b \leftrightarrow ft\} \end{aligned}$$

The construction of a model sharing only the phosphatase is similar. Figure 7.7 shows the behaviour of both models. We can see that indeed the jPTOs still synchronise when coupled less strongly. We can also see that each jPTO, given its own pool of phosphatase, spends more time in the less phosphorylated states as it can dephosphorylate at a greater rate than it can phosphorylate. The reverse is true when given its own pool of kinase. If the concentration of each enzyme was adjusted accordingly, so $c_{F_S} + c_{F_T} = c_E$, then the system behaves as the coupled jPTOs sharing both kinase and phosphatase (as Figure 7.8a).

Figure 7.7 Weaker jPTO coupling, sharing only one enzyme.



(a) Coupling by sharing kinase only. **(b)** Coupling by sharing phosphatase only.

7.3.3 Coupling non-identical clocks

This experiment determines the behaviour observed when coupling a jPTO with cluster 1 parameters (jPTO1) and a jPTO with cluster 2 parameters (jPTO2). The behaviour of the two jPTOs separately is as in Figure 7.3.

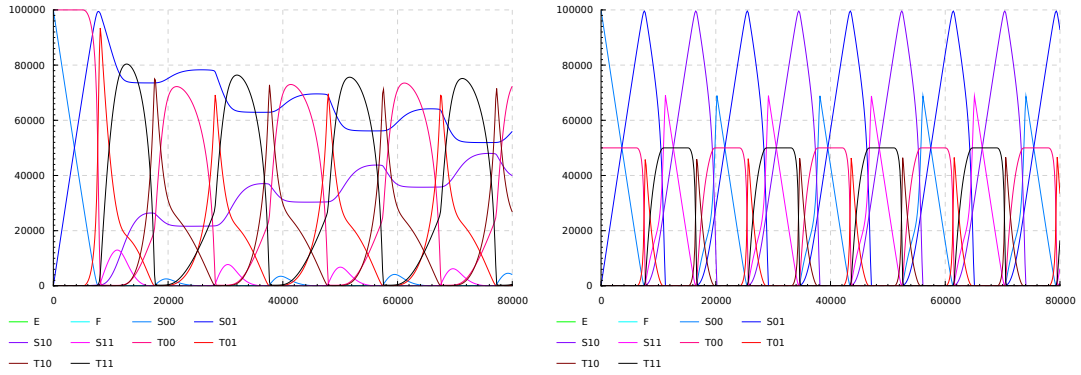
The coupled model is exactly the same in structure as the basic model in Section 7.3.1, except that the rate parameters for the Txx jPTO are changed for the cluster 2 parameters. As in that model, the two jPTOs share a pool both of the enzymes.

The behaviour of this model can be seen in Figure 7.8a. It appears that, after a transient phase, jPTO1 does settle into the familiar oscillation pattern, but jPTO2 shows low amplitude oscillations in comparison. It appears that jPTO2 is dominated by jPTO1. If we reduce the initial concentration of jPTO1 by half (jPTO1/2) then jPTO2 is no longer dominated; this can be seen in Figure 7.8b. Reducing the concentration of jPTO1, we can see that the two jPTO types will indeed synchronise.

7.3.4 Coupling out of phase

This experiment determines the behaviour of coupling a jPTO with an identical jPTO, but out of phase. To achieve this we take two jPTO models of identical structure. The first is the same as jPTO1, in Figure 7.3a. The second is jPTO1 shifted by a quarter of its phase, we call this jPTO1-90 and its behaviour is shown

Figure 7.8 Coupling of two non-identical clocks, sharing an enzyme pool.



(a) jPTO1 + jPTO2. **(b)** jPTO1/2 + jPTO2.

in Figure 7.9a.

The model jPTO1-90 has identical structure to jPTO1, however the initial concentration parameters are altered to reflect the state of the model after a quarter of its oscillation period. The continuous π-calculus software tool supports this construction simply by allowing the user to simulate a model for a given time period and returning a new model whose initial parameters are those of the original model after the given time has elapsed.

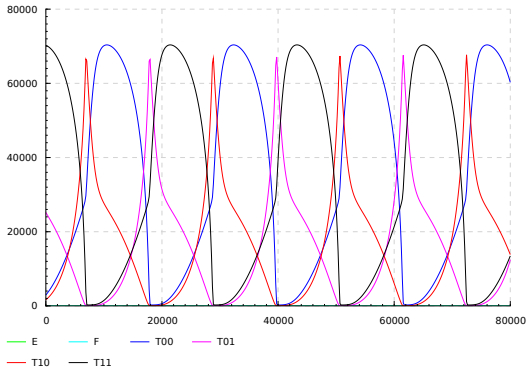
When the two models are composed we see the behaviour in Figure 7.9b. We can see that, after a transient period, the cycles of the two jPTOs synchronise. In a longer run, Figure 7.9c, we can see that the relative concentrations of each of the phosphorylation states of each molecule do indeed synchronise.

For comparison we also coupled jPTO1 with jPTO1s in various phase states. Synchronisation appears to occur when jPTOs are coupled in any phase. This suggests that the synchronisation of two jPTOs is quite robust. Figure 7.9d shows synchronisation when jPTO1 is coupled with a jPTO in anti-phase: jPTO1-180.

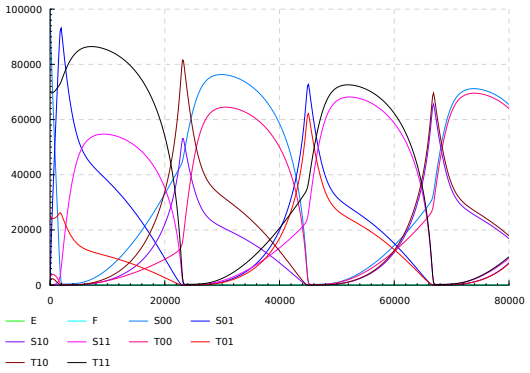
7.3.5 Driving other reactions

Another important function for a circadian oscillator is that it should drive or regulate other processes. In this experiment we determine the behaviour of the jPTO when it is used to regulate a very simple phosphorylation/dephosphorylation reaction.

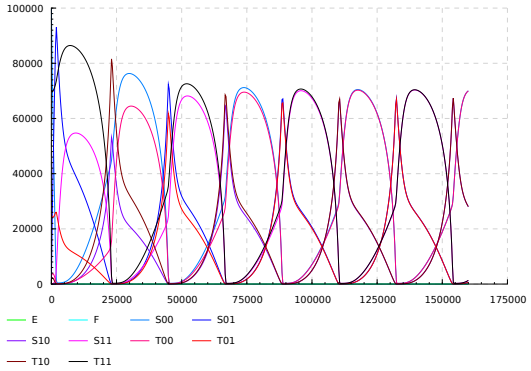
Figure 7.9 Coupling of two identical clocks, sharing an enzyme pool, starting out of phase.



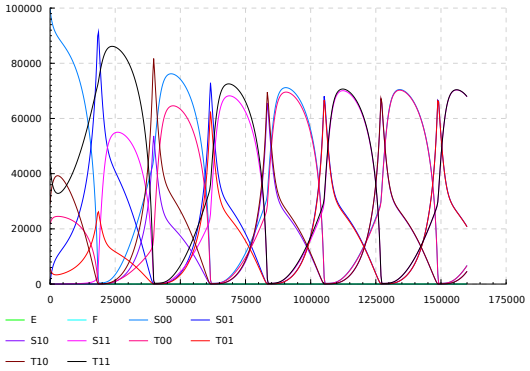
(a) jPTO1-90.



(b) jPTO1 + jPTO1-90.



(c) jPTO1 + jPTO1-90, long run.



(d) jPTO1 + jPTO1-180, long run.

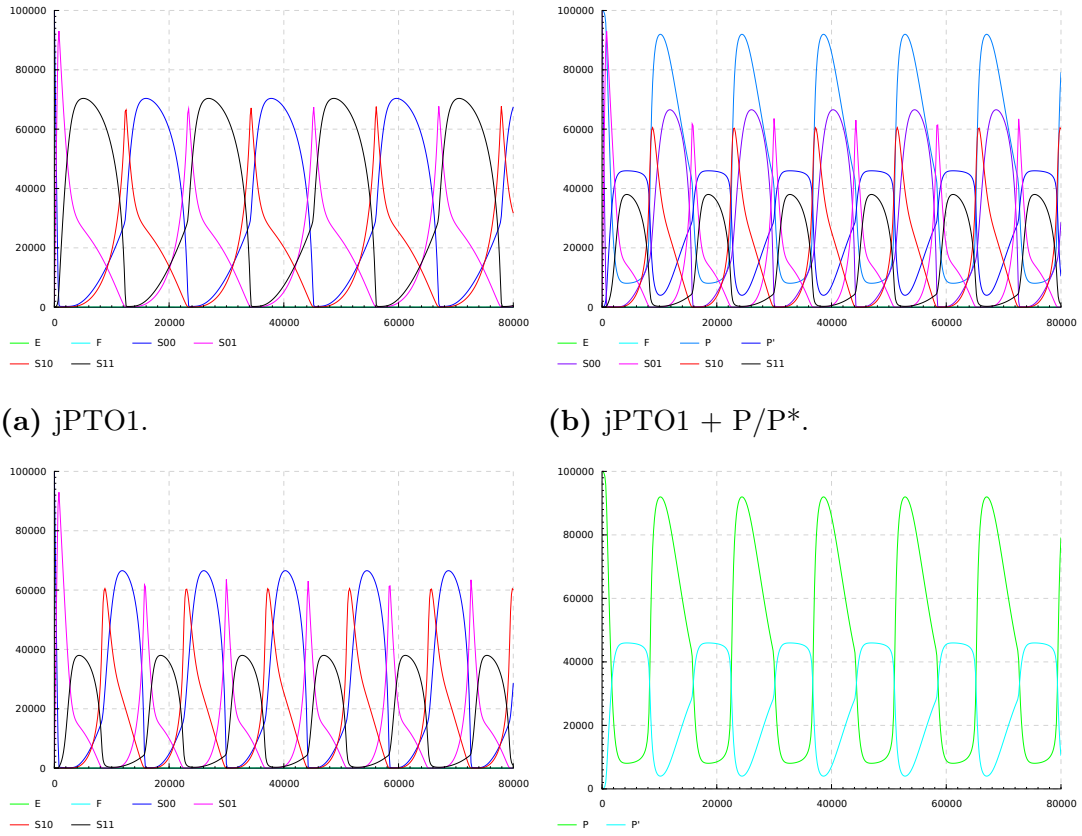
To model this we first construct a model of a molecule, P, which can be phosphorylated to P*. The catalyst for phosphorylation in this case will be S11, the doubly phosphorylated substrate molecule of jPTO1. The molecule P* dephosphorylates autonomously back to P. This model can then simply be coupled with the model jPTO1 and we observe the emergent dynamics. Rate parameters and initial concentrations of the reaction P/P* were chosen to best illustrate this emergent behaviour.

We can see in Figure 7.10b that the jPTO successfully drives the phosphorylation reaction. The dynamics of jPTO1 and P/P* in the coupled model are separated in Figures 7.10c and 7.10d. The dynamics of jPTO1 is shown again in Figure 7.10a for comparison. In Figure 7.10d we can see that the concentrations of each state of P oscillate with the period of the jPTO.

In Figure 7.10c we can see that the jPTO's behaviour is in fact altered by driving

the other reaction. The period is less than that of jPTO1 uncoupled and we can see lower concentrations of S11 appear to be present as some is sequestered by P.

Figure 7.10 Coupling of a jPTO with a simple metabolic reaction pathway.



(a) jPTO1.

(b) jPTO1 + P/P*.

(c) jPTO1 after coupling.

(d) The driven reaction P/P*.

7.3.6 Perturbation

Another useful property of a circadian oscillator is that it is robust to some perturbations—although others may disrupt it. In this experiment we determine the behaviour of the jPTO when perturbed by a pulse of some inhibitor.

To construct a model for this we first construct an inhibitor molecule which rapidly appears in the system and decays rapidly. The mechanism for inhibitor appearing in the system is to have another molecule which is initially present and autonomously becomes the inhibitor. The inhibitor then decays. We will use the inhibitor to bind and sequester components of the jPTO.

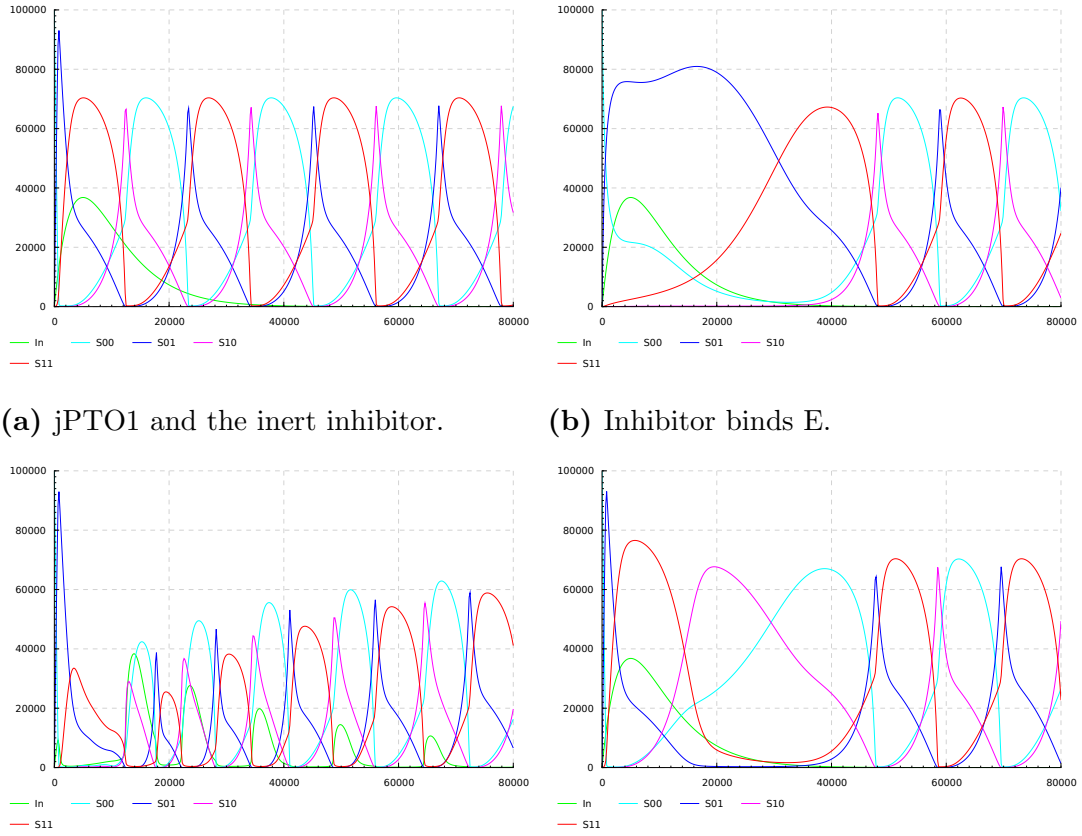
Figure 7.11a shows jPTO1 coupled with the inhibitor; in this case the inhibitor is

inert and does not affect the jPTO. If we then alter the inhibitor so it binds and sequesters various components of the jPTO then we can determine how robust the oscillator is to perturbation.

Figure 7.11 shows the results of the inhibition of each of the components of the jPTO. Figures 7.11b and 7.11d show the effect of inhibiting each of the enzymes E and F. When the enzyme is inhibited there is a transient period—about as long as the pulse—and then the jPTO settles back into its normal oscillation. This shows that the jPTO is robust to temporary sequestration of its enzymes.

Figure 7.11c shows the result of the inhibitor sequestering the doubly phosphorylated substrate molecule S11. Here the fact that S11 itself is only present in pulses and the fact that the inhibitor does not decay when it is bound to S11 means that the inhibitor remains in the system for longer. We can see echo pulses as the inhibitor binds and unbinds the fluctuating concentration of S11. However, overall, the inhibitor eventually decays and the system stabilises. This shows that the substrate is robust to perturbation.

Figure 7.11 Perturbation of a jPTO with a pulse of inhibitor.



(a) jPTO1 and the inert inhibitor.

(b) Inhibitor binds E.

(c) Inhibitor binds S11.

(d) Inhibitor binds F.

7.4 Model checking experiments

The experiments in the previous section show a number of properties which are mostly amenable to analysis by conventional techniques. The compositional nature of cπ models aids greatly in the model construction for models where we are looking at compositions of two models or composition with an inhibitor; something which is much more difficult to do while working directly with ODEs. However the analysis of these models is little more than the inspection of time series for a relatively small set of models and initial conditions.

We will see, in this section that we can use $\mathcal{LBC}$ to automate the process of inspecting time series for a given behaviour. Moreover, and most importantly, we can define higher-order experiments which require many models and many initial conditions. We gain a means to express a set of computational experiments, which in a conventional setting would require case-specific programming, and

to have them automatically checked. That is, the model checker provides the implementation which would have been done by hand previously.

In Sections 7.4.1 to 7.4.3 we discuss some general ways to formulate relevant questions in $\mathcal{LBC}$. Then in Section 7.4.4 we apply these formulae to our model and examine the results.

7.4.1 Composition properties

$\mathcal{LBC}$ takes advantage of model compositionality. Specifications about the behaviour of a model when it is composed with another model can be made using the context modality. We can make use of this in analysing the behaviour of coupled oscillators.

For example: $PTO1 \models PTO2 \triangleright \phi$ states that when we couple two PTOs we have some behaviour ϕ. Likewise we could state $PTO \models Inhib \triangleright \phi$ meaning that our PTO has some behaviour ϕ when we introduce an inhibitor ($Inhib$).

However, the most interesting properties are those which make a statement about introducing something over time. For example: $PTO1 \models \mathbf{G}_t(PTO2 \triangleright \phi)$ which states that if we couple $PTO2$ with $PTO1$ at any time until t then we have some behaviour ϕ.

7.4.2 Complex dynamics

$\mathcal{LBC}$ also has the power to express complex dynamics, such as periodicity and oscillation. Numerous bodies of work have attempted to express oscillation properties in standard temporal logic [10, 9, 23], but all fall short of a general formula for oscillation. It is possible to express oscillation, however, with some prior knowledge of the type of oscillation. Following the idea in Calzone et al. [23] and extending it to a time-bounded logic, we can express oscillation in the temporal fragment of $\mathcal{LBC}$ as follows:

$$PTO \models \mathbf{G}_{[0,t]}(\mathbf{F}_{[0,p]}(([S]' > 0) \wedge \mathbf{F}_{[0,p]}([S]' < 0)))$$

where $[S]'$ is the first derivative of $[S]$ with respect to time. The formula states

that at any time up to t the concentration of S will, within a further time p, be rising and then within another additional time p be falling. This describes a repeated rising and falling with period at most p. Whilst this is not a general formula, it does cover a large class of sustained oscillation. However, its weakness is that it does not distinguish from noise—although noise is not a problem when studying ODE models.

It has been shown that more expressive logics can express more general formulae for oscillation; for example Dluhoš et al. [39] show that one can use a "freeze operator" to do this. In fact, it is possible to give a general formula for sustained—and not necessarily regular—limit cycle periodicity using $\mathcal{LBC}$. The formula:

$$PTO \models \mathbf{F}_{[p_{min},p_{max}]}(\widehat{PTO} \triangleright (\mathbf{F}_{[0,s]}\mathbf{G}_{[0,t]}(|[S] - [\widehat{S}]| < \epsilon))) \tag{7.1}$$

where $\widehat{PTO}$ is a copy of PTO, S is the species being observed, $\widehat{S}$ is the copy of S in $\widehat{PTO}$, and s is a maximum transient period before reaching the limit cycle. The formula states that if we introduce $\widehat{PTO}$ after some period in $[p_{min}, p_{max}]$ then, within s, $[S]$ and $[\widehat{S}]$ will synchronise to within ϵ for at least time t. This essentially takes a copy of the model, shifts it forward in time by $p_{min} \leqslant t \leqslant p_{max}$, and determines if it matches up with the original model. If it does, allowing for some initial transient period, then the model is periodic in species S.

In the context of our case study, we can now check if coupled PTOs still oscillate:

$$PTO1 \models \mathbf{G}_{[0,c]}(PTO2 \triangleright \texttt{Osc})$$

where c is the end of the first cycle of $PTO1$ and `Osc` is one of our oscillation formulae from above. If coupling the PTOs at any time within the first cycle of $PTO1$ gives a system which still oscillates—with some period bounds, as above—then the formula will be true.

7.4.3 Perturbation response

$\mathcal{LBC}$ can be used to express properties of a system under perturbation. For example, one might wish to determine if some perturbation causes a greater peak concentration in a species S. The formula:

$$PTO \models \mathbf{F}_{[0,t]}(P \triangleright \mathbf{F}_{[0,r]}([S] > pk))$$

states that some peak value pk is exceeded under some perturbation P, within time t, where r is the maximum expected time of the peak after the perturbation. As the perturbation P could be any model, it could simply be a quantity of some species, a constant amount of inhibitor, a pulse of inhibitor, etc.

Of particular interest in the study of oscillators are the *phase response* [51] characteristics of system. That is, given a short perturbation, at any point in the cycle, what is the effect on the phase of the oscillation? Biologists often plot a *phase response curve*, using a large number of experiments, to visualise the phase response. $\mathcal{LBC}$ cannot give such a precise and quantitative account of phase response as this, however it is certainly possible to formulate some more qualitative—or even semi-quantitative—properties of phase response. For example:

$$PTO \models \widehat{PTO} \triangleright \mathbf{F}_{[c_1,c_2]}(P \triangleright (\mathbf{G}_{[t_1,t_2]}([\widehat{S}]' > 0 \implies \mathbf{F}_{[s_1,s_2]}[S]' > 0)))$$

states that some perturbation P applied within $[c_1, c_2]$ will cause a *forward* phase shift in $[s_1, s_2]$. t_1 is a known max transient period after introducing P, t_2 is a sensible maximum time to simulate for, and the formula assumes that we know the perturbed system still oscillates.

7.4.4 Results

The following results of verifying the above $\mathcal{LBC}$ properties against the cπ models of Jolley's PTO were obtained by using the reference implementation of the $\mathcal{LBC}$ signal-based model checker—details of which are in Chapter 8. First we show a number of formulae which give the same results as the experiments performed above, albeit without the need to manually inspect a simulation trace. These results serve to verify the use of the model checker. Finally we show the results of checking formulae which describe higher-order computational experiments, i.e. those which check properties which would require the manual inspection of multiple simulation runs.

7.4.4.1 Oscillation

Our first test was to check for oscillation using Formula 7.1. We let:

$$\texttt{Osc} = \mathbf{F}_{[p_{min},p_{max}]}(\widehat{jPTO} \triangleright (\mathbf{F}_{[0,s]}\mathbf{G}_{[0,t]}(|[S00] - [\widehat{S00}]| < \epsilon)))$$

where: we know that the period is around 24000 so we set $p_{min} = 23000$ and $p_{min} = 25000$; we know the system will reach limit cycle within $s = 10000$; we must choose an oscillating species and so choose $S = S00$; a reasonable time to simulate for is $t = 80000$—a few cycles; and we choose $\epsilon = 1$ as our concentration accuracy. The copy model $\widehat{jPTO}$ can be constructed in the same manner as the copy model in Section 7.3.1 or by using the appropriate function in the reference implementation.

Upon checking $jPTO1 \models \texttt{Osc}$ and $jPTO2 \models \texttt{Osc}$ we find that both return *True*. This confirms what we have been able to determine manually from inspecting the simulation traces in Figure 7.3. Moreover, it shows that the $\mathcal{LBC}$ formula is a succinct and precise means of expressing the oscillation property and the model checker provides an automatic means for testing such a hypothesis.

7.4.4.2 Coupled oscillators

The next step is to test coupled oscillators for oscillation, as in Section 7.3.1. First we take identical PTOs with Cluster 1 parameters: *jPTO1a* and *jPTO1b*. Upon checking $jPTO1a \models jPTO1b \triangleright \texttt{Osc}$ using the same formula parameters as above, we find that the result is *False*. This is because, as seen in Figure 7.6a, the period of the coupled oscillators is doubled. Therefore, upon relaxing the desired period range to $p_{min} = 23000$ and $p_{max} = 49000$ we find the formula is satisfied and the checker returns *True*.

Once again, this result is a validation of the model checker and shows the power of $\mathcal{LBC}$ to express composed model properties of the model checker to test such a hypothesis.

7.4.4.3 Out-of-phase coupling

We can now begin to demonstrate the higher-order capabilities of the model checker. In Section 7.3.4 we showed that for a limited number of out-of-phase couplings of *jPTO1a* and *jPTO1b* the two systems did indeed synchronise and oscillate together after an initial transient period. This however does not confirm whether this is the case for *all* phase shifts.

Using the test $jPTO1a \models \mathbf{G}_{[0,c]}(jPTO1b \triangleright \texttt{Osc})$ we can use the model checker to

give a greater guarantee that coupling the oscillators in any phase shift, up to c times the length of one cycle. Here we know that the length of one cycle is no more than, say, $c = 26000$.

Upon checking, again with the above formula parameters and the relaxed period range, we find that the result is *False*. This is because we have not accounted for the lengthened transient period when coupling out of phase. If we increase the parameter s to 120000 we find the formula is now satisfied, the result is *True*. This gives a much stronger guarantee that all out-of-phase couplings oscillate than a limited number of manually inspected simulation traces would give.

7.4.4.4 Phase response

Another higher-order property is the phase response characteristic. Using the inhibitor pulse model in Section 7.3.6—except using a pulse which lasts for fewer than 2000 time units—and the formula in Section 7.4.3 we can place some bounds on the phase response characteristics of the model:

$$jPTO1 \models \widehat{jPTO1} \triangleright \mathbf{F}_{[c_1,c_2]}(Pulse \triangleright (\mathbf{G}_{[t_1,t_2]}([\widehat{S00}]' > 0 \implies \mathbf{F}_{[s_1,s_2]}[S00]' > 0)))$$

where: $[c_1, c_2] = [10000, 34000]$ which is roughly one cycle, this limits the computation; the maximum expected transient period is $t_1 = 10000$; the maximum time to compare oscillations is $t_2 = 80000$; and $[s_1, s_2] = [0, 1000]$ ensures that the whole formula states that: "there is always a forward phase response of no more than 1000 time units".

The model checker confirms that this statement is true for this model. So our small pulse may delay the cycle, but only by a relatively small time; it does not speed up the cycle.

7.5 Conclusions

In this chapter we have shown that, using a combination of cπ and $\mathcal{LBC}$, we can express a variety of interesting and complex properties of biochemical models. We have shown that precise and succinct statements of complex properties can be

built up in a modular fashion. One can even think of "higher-order" $\mathcal{LBC}$ properties as precise and succinct statements of an experiment, or set of experiments, to be carried out by the model checker.

Whilst the current reference implementation of the model checker may take a number of hours to check a complex property, such as the phase response formula, the time to check such properties is not prohibitively long in real terms. Indeed, when one considers the time required to construct a manual implementation of such experiments, it is in some ways less costly to task the machine with this, say overnight. One might also consider the time it would take to carry out these experiments in a wet lab and to collect and analyse the data.

Furthermore, an implementation of the model checker using sensitivity analysis, from Chapter 9, has the potential to speed up this process. There is also substantial potential for speeding up the implementation by using a C-based ODE library rather then GNU Octave.

Chapter 8

Tool implementation

In this chapter we give an overview of the reference implementation of cπ and $\mathcal{LBC}$. The Continuous Pi-calculus Workbench (CPiWB) toolset has been implemented in Haskell[1] [73], using GNU Octave[2] [45] for the numerical computation and solution of ODEs. In Section 8.1 we describe the modular structure of the CPiWB library and its use in the main user-interface program. In Section 8.2 we give a brief overview of the command-driven user interface and its capabilities. Finally, in Section 8.3, we describe how the tool was used for systematic testing and profiling of the $\mathcal{LBC}$ model checker; this section aims to allow a third party to replicate the experimental results of this thesis using the CPiWB.

The tool is open source software, freely available under the GNU General Public License[3], and available to download from: `http://github.com/chrisbanks/cpiwb`.

8.1 Framework

We implemented the CPiWB as a modular set of libraries, in Haskell, with an executable user interface. Sound software engineering practice dictates that complex applications should be built in a modular fashion, reducing code duplication, increasing the re-usability of utility functions, separating concerns, simplifying

[1]The Haskell Programming language: `http://www.haskell.org/`
[2]GNU Octave: `http://www.gnu.org/software/octave/`
[3]GNU General Public License: `http://www.gnu.org/licenses/gpl.html`

maintenance, and allowing third parties to build on the library without knowing the implementation details. The use of Haskell allowed us to make use of call-by-need (lazy) evaluation for the short-circuiting property of the model-checking algorithms in Chapter 5.

The CPiWB makes use of both internal and external ODE solver libraries. The internal solvers make use of Haskell libraries (see below) and the external solver is a call to GNU Octave.

The CPiWB toolset includes: a full interactive interpreter and compiler for cπ; an implementation of $\mathcal{LBC}$ and its various model-checking algorithms; internal and external ODE solving; analysis tools for inspecting model structure, plotting time series and phase plots, and exporting data; and other tools for manipulating cπ models.

The following is a brief description of each of the library modules:

CPi/Lib.hs contains a number of utility functions which are used throughout the other modules.

CPi/Semantics.hs contains the implementation of cπ, its data structures, and related functions.

CPi/Logic.hs contains the implementation of $\mathcal{LBC}$ its data structures, and related functions. This includes the implementation of the trace-based model-checking algorithms.

CPi/Parser.hs contains the parsers for cπ model descriptions and $\mathcal{LBC}$ formulae. We use the Parsec[4] parser combinator library to achieve this.

CPi/ODE.hs contains the implementation of the ODE extraction algorithm for cπ. This also contains the internal ODE solvers, implemented using the HMatrix[5] package.

CPi/Matlab.hs converts the internal ODE representation to a GNU Octave script and calls Octave to solve the ODEs, retrieving the result. This improves support for complex/stiff systems.

CPi/Plot.hs allows the plotting of time series for analysis, using the graphical

[4]Parsec parser combinators library: `http://hackage.haskell.org/package/parsec`
[5]HMatrix linear algebra library: `http://hackage.haskell.org/package/hmatrix`

Chart-GTK[6] Haskell package.

CPi/Signals.hs implements the signal-based model-checking algorithm.

8.2 User interface

The CPiWB command-driven user interface provides an environment for defining or importing cπ models, solving, analysing, and performing $\mathcal{LBC}$ model checking. The interface is text-based, similar to other mathematical software packages such as GNU Octave and the Edinburgh Concurrency Workbench[7]. The following is a brief description of some of the interface commands:

help gives an overview of all commands and provides detailed help on each command.

load loads an existing cπ model into the environment.

species/process defines a new species/process.

trans allows the user to inspect the static structure of the cπ model.

odes compiles the cπ model to a set of ODEs.

plot compiles, solves, and plots the time series of a cπ model.

check checks that a cπ model satisfies an $\mathcal{LBC}$ formula.

evolve takes a cπ process and returns the process corresponding to the conditions after a specified model runtime.

Figure 8.1 shows a typical cπ model, a snapshot of the CPiWB workflow, and an example execution of the model.

8.3 Testing and profiling

All of the cπ models, ODEs, traces, $\mathcal{LBC}$ checks, etc. presented in the thesis were done using this toolset. We created a testing and profiling suite for the $\mathcal{LBC}$ model checker for the purpose of systematically testing and determining the

[6]Chart-GTK graphing library: `http://hackage.haskell.org/package/Chart-gtk`
[7]Edinburgh Concurrency Workbench: http://homepages.inf.ed.ac.uk/perdita/cwb/

performance characteristics of the various model-checking algorithms which were developed. The suite consists of a test model, a set of systematically chosen $\mathcal{LBC}$ formulae, and a set of test scripts which profile the performance of checking each formula against the test model.

The test model is that which appears in Appendix A and the test formulae are those which are detailed in Appendix B. The formulae are chosen specifically to give a systematic cross-section of possible $\mathcal{LBC}$ formulae. The formulae are chosen to expose the characteristics of each model-checking algorithm; for example, they range over increasing complexity of formulae and each of the chosen atomic propositions are more or less amenable to short-circuiting.

The test scripts make use of the GHC profiling libraries and tools[8] to test the performance of checking each formula against the model and to output the results. The same model, set of formulae, and test scripts on the same machine were used to profile the performance of each algorithm to allow direct comparison. The test script sources are available in the software repository[9].

The tool has also been tested by building and analysing a number of models of varying size and complexity. The models range from simple, contrived examples to large and well-known biological models. All of the models appearing in this thesis and Kwiatkowski's thesis [62] have been built and analysed using this software.

[8]GHC profiling tools: `https://www.haskell.org/ghc/docs/7.6.3/html/users_guide/profiling.html`

[9]CPiWB: `https://github.com/chrisbanks/cpiwb`

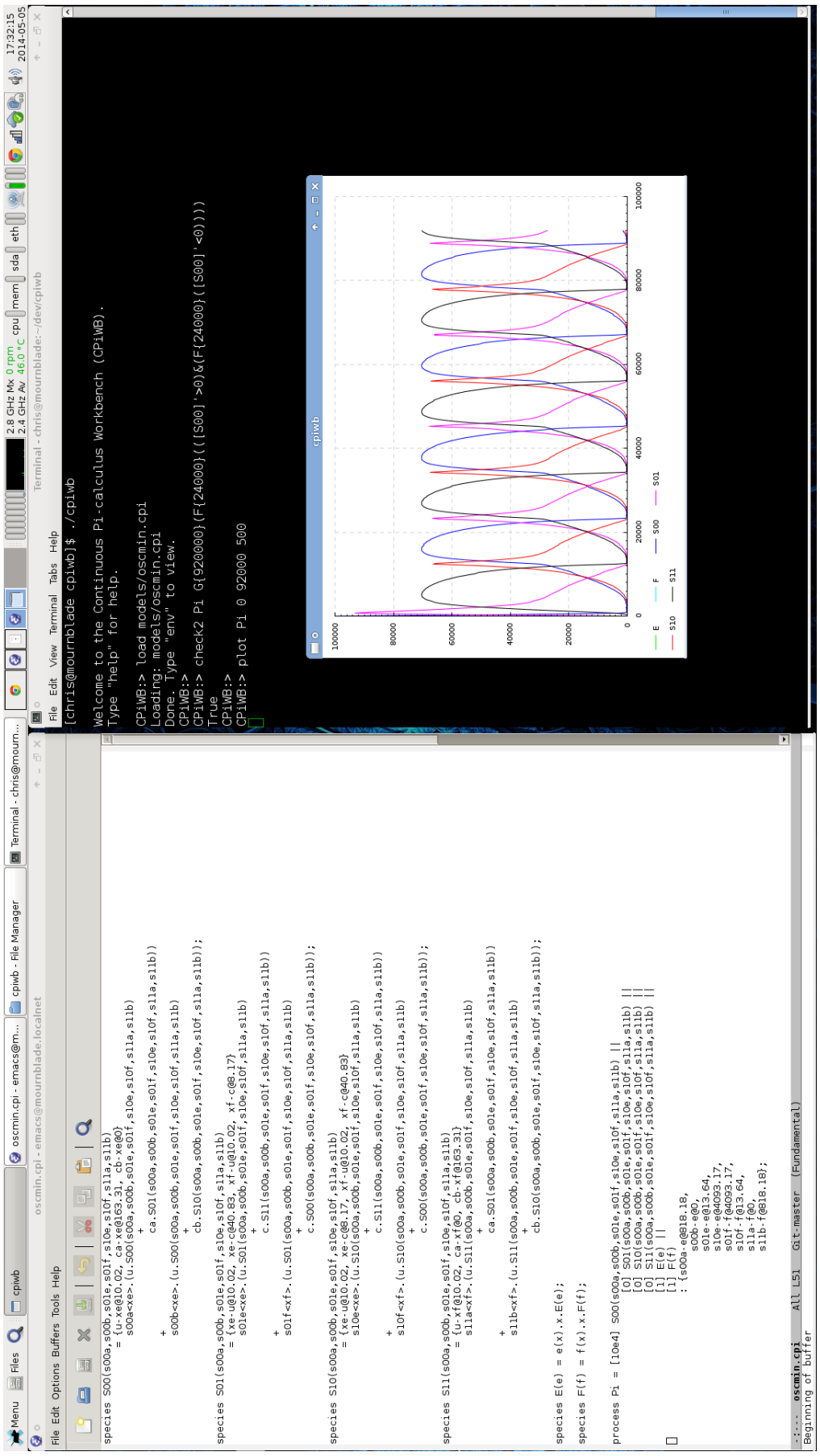


Figure 8.1: The CPiWB user interface: (*left*) cπ code for PTO model; (*right*) CPiWB interface and plot output.

Chapter 9

Sensitivity analysis

In Chapter 6 we introduced the method of using Boolean signals to compress the simulation trace for more efficient model checking. Here we attempt to improve this by defining a method by which we could potentially reduce the number of calls to the ODE solver in practice, whilst also improving the coverage of context introduction between the original sample points.

It must be understood that this chapter represents only the foundation of a method and a proposal for further work. The ideas herein have not, at the time of writing, been implemented or tested. However, we do believe our conjecture is well-reasoned and that the method has the potential to show some improvement over the previous methods. We do not present this chapter in the same light as the previous ones, we merely present the ideas and propose the testing of the above conjecture as future work.

The key to the method comes from a study of sensitivity analysis for safety properties by Donzé and Maler [42]. Sensitivity analysis is used to systematically check a system with uncertain initial conditions; the use of sensitivity analysis ensures that a few discrete simulations cover a continuous space of initial conditions. Here we can apply the same principle to check $P^t \vDash Q \triangleright \phi$ for a range of t, considering $P^t \parallel Q$ over this range as a space of initial conditions for which to check ϕ. This initial condition space is continuous, but it is possible to find a finite set of regions which is sufficient to cover this space using an adaptation of Donzé and Maler's technique; thus potentially reducing the number of calls made to the ODE solver.

9.1 Expansion function

Our approach is based on analysing, together, the set of trajectories of a dynamical system arising from a set X_0 of initial conditions. For any $x_0 \in X_0$, define $\xi_{x_0}(t)$ to be the corresponding trajectory from initial state x_0. Define $\texttt{reach}_{\leqslant t}(X_0)$ as the set of states reachable within time t from some initial state in X_0, with $\texttt{reach}_{=t}(X_0)$ as those states reachable in exactly time t. Suppose now that we have a distance metric d between points, and extend it to a distance between points and sets:

$$d(x, X) = \inf_{y \in X}(d(x, y))$$

The set $\mathcal{B}_\delta(x)$ is the δ-ball around point x and the set $\mathcal{B}_\delta(X)$ is the δ-ball around set X thus:

$$\mathcal{B}_\delta(X) = \bigcup_{x \in X} \mathcal{B}_\delta(x)$$

Donzé and Maler show that it is possible to find the ball which tightly over-approximates $\texttt{reach}_{=t}(\mathcal{B}_\delta(x_0))$ by means of the *expansion function*. Therefore it is possible to construct a "flow tube" around a trajectory which tightly over-approximates the reachable set.

Definition 9.1.1 (Expansion function [42])**.** For $x_0 \in X_0$ and some $\epsilon > 0$ we define the *expansion function* $\Xi_{x_0,\epsilon}$ of trajectory ξ_{x_0} to be the map taking any time $t \geqslant 0$ to the smallest $\delta \geqslant 0$ such that all trajectories with initial state in $\mathcal{B}_\epsilon(x_0)$ reach a point in $\mathcal{B}_\delta(\xi_{x_0}(t))$ at time t:

$$\Xi_{x_0,\epsilon}(t) = \sup_{d(x_0,x) \leqslant \epsilon} d(\xi_{x_0}(t), \xi_x(t))$$

The value of the expansion function is the radius of the tightest ball around the reachable set from the ϵ-ball around the initial condition. The key here is that if we take the initial set to be the ball which tightly bounds our possible initial conditions then we can compute an over-approximation of the reachable set and therefore prove that we do not reach a state where some ϕ holds. Donzé and Maler [42] show that $\Xi_{x_0,\epsilon}(t)$ can be computed via *sensitivity to initial conditions* $\frac{\partial \xi_{x_0}}{\partial x_0}(t)$, which is commonly implemented by numerical solvers. They show also that the error in the numerical approximation is quadratic in ϵ. Accordingly, we

propose to limit the size of the balls used to some chosen parameter θ. Where the ball around a set of initial conditions is larger then we split it into a covering set of smaller balls.

Donzé and Maler then have a scheme for refining the over-approximation for an arbitrary set of initial conditions in $\mathbb{R}^n$. However here our initial conditions are less general and so, in Section 9.2, we give a more specific method used for computing the signal of a context modality.

9.2 Application to checking $\mathcal{LBC}$

We propose to apply expansion functions to the model checking of $\mathcal{LBC}$ formulae. Specifically, we extend the signal-based algorithms of Chapter 6 to signals along *flow tubes*: the set of all trajectories from a ball of initial conditions. This also requires us to extend our algorithm to three-valued logic, recognising that an $\mathcal{LBC}$ formula may not have the same value across a ball.

We can apply the principle of computing traces of flow tubes, where necessary, instead of traces of trajectories as follows. First we set out some preliminary definitions.

The set B is the set of balls in $\mathbb{P}$ and for $\beta \in B$: $c(\beta)$ is the centre point of the ball and $r(\beta)$ is the radius. The set *Tube* is the set of flow tube traces in $\mathbb{P}$ where a flow tube trace is of the form:

$$(t_0, \beta_0), \ldots, (t_n, \beta_n)$$

where $t_i \in \mathbb{R}^+$ is a time point and $\beta_i \in B$. The ball $\beta \parallel P$ is the ball β translated by the process vector P.

Definition 9.2.1 (Trace and flow tube trace)**.** We have a function:

$$\texttt{trace} : \mathbb{P} \times \mathbb{R}^+ \times \mathbb{R}^+ \to \mathit{Trace}$$

where $\texttt{trace}(P, t, \rho)$ gives the trace of process P up to time t with resolution ρ using numerical simulation. We also have a function:

$$\texttt{tube} : B \times \mathbb{R}^+ \times \mathbb{R}^+ \to \mathit{Tube}$$

where $\texttt{tube}(\beta, t, \rho)$ gives the flow tube trace from the ball β for time t with resolution ρ; each ball in the flow tube trace is given by $\Xi_{c(\beta),r(\beta)}(t_i)$ for each t_i up to t such that $t_{i+1} - t_i = \rho$.

9.2.1 Three-valued signals

The set $\mathbb{B}_\perp$ is $\{\mathit{True}, \mathit{False}, \perp\}$ where $\perp$ is of "uncertain" Boolean value. The set $\mathit{Signal}_\perp$ is the set of finite length $\mathbb{B}_\perp$ signals, signals with uncertainty, defined as follows:

Definition 9.2.2 (Finite length $\mathbb{B}_\perp$ signal)**.** A finite length $\mathbb{B}_\perp$ signal s of length r is a function $s : [0, r) \to \mathbb{B}_\perp$. A finite length $\mathbb{B}_\perp$ signal has finite variability and, therefore, may be represented by a finite interval covering. The interval covering, minimal covering, and definitions of consistency and refinement are the same as in Definition 6.1.1.

The set of *positive* intervals of s is $\mathcal{I}_s^+ = \{I \in \mathcal{I}_s : s(I) = \mathit{True}\}$, the set of *uncertain* intervals of s is $\mathcal{I}_s^\perp = \{I \in \mathcal{I}_s : s(I) = \perp\}$, and the set of *negative* intervals is $\mathcal{I}_s^- = \mathcal{I}_s \setminus (\mathcal{I}_s^+ \cup \mathcal{I}_s^\perp)$.

9.2.2 Three-valued signal combinators

We can now define the signal combinators for three valued signals. Note that we once again overload the logical operators to apply to Boolean signals and three-valued signals.

Definition 9.2.3 ($\mathbb{B}_\perp$ signal combinators)**.** The signal combinators apply the logical connectives $(\neg, \wedge)$ and temporal modalities $(\mathbf{F}, \mathbf{U})$ to signals (s_ϕ, s_ψ) and are defined as follows:

$\neg s_\phi$

: Negation is a simple negation of the signal such that $\mathcal{I}_{s_{\neg\phi}}^+ = \mathcal{I}_{s_\phi}^-$ and $\mathcal{I}_{s_{\neg\phi}}^\perp = \mathcal{I}_{s_\phi}^\perp$.

$s_\phi \wedge s_\psi$

: For conjunction we first compute a refinement of the coverings $\mathcal{I}_\phi^R \prec \mathcal{I}_\phi$ and $\mathcal{I}_\psi^R \prec \mathcal{I}_\psi$ such that $\mathcal{I}_\phi^R = \mathcal{I}_\psi^R$ and is the sequence of intervals $I_1^R, \ldots, I_n^R$.

The conjunction is then computed interval-wise such that $s_{\phi\wedge\psi} = s_\phi \wedge s_\psi$ where $s_{\phi\wedge\psi} = \bot$ when either $s_\phi = \bot$ or $s_\psi = \bot$. The minimal covering $\mathcal{I}_{s_{\phi\wedge\psi}}$ is then computed by merging any adjacent intervals of the same value.

$\mathbf{F}_{[a,b]} s_\phi$

The temporal $\mathbf{F}_{[a,b]}$ modality is computed by back-shifting the positive and uncertain intervals. $\mathcal{I}^+_{\mathbf{F}_{[a,b]}\phi}$ is computed by taking each interval $I \in \mathcal{I}^+_\phi$ and computing its back-shifting $I \ominus [a,b] \cap \mathbb{R}^+$. In the same way $\mathcal{I}^\perp_{\mathbf{F}_{[a,b]}\phi}$ is computed and where any interval overlaps with an interval in $\mathcal{I}^+_{\mathbf{F}_{[a,b]}\phi}$ the overlapping portion of the interval is removed from $\mathcal{I}^\perp_{\mathbf{F}_{[a,b]}\phi}$. The minimal covering $\mathcal{I}_{\mathbf{F}_{[a,b]}\phi}$ is then computed by merging any adjacent intervals of the same Boolean value.

$s_\phi \mathbf{U}_{[a,b]} s_\psi$

The fundamental temporal $\mathbf{U}_{[a,b]}$ modality can be computed on the basis that $\phi \mathbf{U}_{[a,b]} \psi \iff \phi \wedge \mathbf{F}_{[a,b]}(\phi \wedge \psi)$ when s_ϕ is a unitary signal. A signal s is unitary if $\mathcal{I}^+_s \cup \mathcal{I}^\perp_s$ is a singleton. So if s_ϕ is unitary and it has a value at t_1 and t_2 then it must hold that value for the whole interval $[t_1, t_2]$. For the case where s_ϕ is not unitary we can decompose it into a set of unitary signals $\{s^1_\phi, \ldots, s^n_\phi\}$ and compute, for each $i \in [1, n]$:

$$s^i_{\phi \mathbf{U}_{[a,b]} \psi} = s^i_\phi \wedge \mathbf{F}_{[a,b]}(s^i_\phi \wedge s_\psi)$$

The signal is then recomposed to give:

$$s_{\phi \mathbf{U}_{[a,b]} \psi} = \bigvee_{i=1}^{n} s^i_{\phi \mathbf{U}_{[a,b]} \psi}$$

Note that we no longer define signal combinators for the context modality signal. The computation of a context modality signal is handled by the model-checking functions defined in the next section.

9.2.3 Model-checking functions

Our sensitive model checker for $\mathcal{LBC}$ over cπ processes is defined by four mutually recursive functions. Function `sat` is the top-level function which computes whether or not a process satisfies an $\mathcal{LBC}$ formula. Function `satB` computes whether a ball satisfies a formula: giving *True*, *False* or $\bot$ according to whether

every point of the ball satisfies the formula, none do, or only some. The function `signal` computes the formula satisfaction signal along a process trajectory. Finally, `signalT` computes a satisfaction signal along a flow tube trace. As with `satB`, this signal is three-valued according to whether the cross-section of the flow tube at each trace instant lies within, outside, or partially within the region satisfying the formula.

The algorithms for our sensitive model checker have two control parameters: a time resolution ρ as the step size for traces and flow tube traces; and θ, the ball radius within which we use the expansion function to extrapolate a flow tube.

Definition 9.2.4 (`sat`)**.** Function $\texttt{sat} : \mathbb{P} \times \Phi \to \mathbb{B}$ is computed recursively as follows:

$$\begin{aligned}
\texttt{sat}(P, \mathit{Atom}) &= \mathit{Atom} \in \mathit{Props}(P) \\
\texttt{sat}(P, \phi \wedge \psi) &= \texttt{sat}(P, \phi) \wedge \texttt{sat}(P, \psi) \\
\texttt{sat}(P, \neg\phi) &= \neg(\texttt{sat}(P, \phi)) \\
\texttt{sat}(P, (Q \triangleright \phi)) &= \texttt{sat}(P \parallel Q, \phi) \\
\texttt{sat}(P, \phi \mathbf{U}_{[a,b]} \psi) &= (\texttt{signal}(P, |\phi \mathbf{U}_{[a,b]} \psi|, \phi \mathbf{U}_{[a,b]} \psi))(0) \ .
\end{aligned}$$

Satisfaction of a non-temporal formula is straightforward and can be determined directly from the initial conditions. Even a context formula, if its subformula is non-temporal, requires only a process composition and an inspection of initial conditions. However, for a temporal formula we compute the signal of its satisfaction over time and then take the initial value of that signal. This leads us to the next function required.

Definition 9.2.5 (`signal`)**.** Function $\texttt{signal} : \mathbb{P} \times \mathbb{R}^+ \times \Phi \to \mathit{Signal}$ is computed recursively as follows, using the Boolean signal combinators from Definition 6.1.2:

$$\begin{aligned}
\texttt{signal}(P, t, \mathit{Atom}) &= \texttt{basicSignal}(P, t, \mathit{Atom}) \\
\texttt{signal}(P, t, \phi \wedge \psi) &= \texttt{signal}(P, t, \phi) \wedge \texttt{signal}(P, t, \psi) \\
\texttt{signal}(P, t, \neg\phi) &= \neg(\texttt{signal}(P, t, \phi)) \\
\texttt{signal}(P, t, \phi \mathbf{U}_{[a,b]} \psi) &= \texttt{signal}(P, t, \phi) \mathbf{U}_{[a,b]} \texttt{signal}(P, t, \psi) \\
\texttt{signal}(P, t, Q \triangleright \phi) &= \texttt{contextSignal}(P, t, Q, \phi)
\end{aligned}$$

where `basicSignal` and `contextSignal` are defined below.

`basicSignal` : $\mathbb{P} \times \mathbb{R}^+ \times \mathit{Atomic} \to \mathit{Signal}$

Here $\texttt{basicSignal}(P, t, \mathit{Atom})$ gives the finite signal s such that for each $(t_i, \vec{c_i}) \in \texttt{trace}(P, t, \rho)$:

$$s([t_i, t_i + \rho)) = (\mathit{Atom} \in \mathit{Props}(\vec{c_i})) \ .$$

`contextSignal` : $\mathbb{P} \times \mathbb{R}^+ \times \mathbb{P} \times \Phi \to \mathit{Signal}$

Function $\texttt{contextSignal}(P, t, Q, \phi)$ computes a signal s as follows. Take set X of all process states in the trace $\tau = \texttt{trace}(P, t, \rho)$, let β_X be the minimum bounding ball around X, and let I be the interval $[t_0, t_n + \rho)$ including time points $t_0, \ldots, t_n$ of τ. If $\texttt{satB}(\beta_X \parallel Q, \phi)$ is either *True* or *False* then

$$s(I) = \texttt{satB}(\beta_X \parallel Q, \phi)$$

defines our signal s. If not, and τ contains only one time point $(t, \vec{c})$, then take

$$s(I) = \texttt{sat}(\vec{c} \parallel Q, \phi) \ .$$

Finally, if `satB` gives $\perp$ and τ contains multiple points, then bisect τ and repeat the procedure for each new τ, X and I.

The signal for an atomic proposition is computed directly from the simulation trace, interpolating between time points. The signal for non-atomic formulae other than context modalities is computed by using the appropriate signal combinators.

The difficult case is for context modalities. For this we compute a trace as for atomic propositions, translate it by the context Q, and then test a bounding ball around all points in the trace. If this is inconclusive we repeatedly refine until we find a set of balls which give a conclusive result. In the worst case this means checking individual points of the trace—as we did in our earlier methods. However, in any other case, we may save computation by checking a whole set of points together in a single ball.

That, however, requires a function to compute satisfaction across a ball.

Definition 9.2.6 (`satB`)**.** The three-valued function $\texttt{satB} : B \times \Phi \to \mathbb{B}_\perp$ for

satisfiability of a formula across a ball is computed recursively:

$$\texttt{satB}(\beta, \mathit{Atom}) = \begin{cases} \mathit{True} & \beta \subseteq \{x \in \mathbb{P} \mid \mathit{Atom} \in \mathit{Props}(x)\} \\ \mathit{False} & \beta \cap \{x \in \mathbb{P} \mid \mathit{Atom} \in \mathit{Props}(x)\} = \emptyset \\ \bot & \text{otherwise} \end{cases}$$

$$\texttt{satB}(\beta, \phi \wedge \psi) = \begin{cases} \mathit{True} & (\texttt{satB}(\beta, \phi) = \mathit{True}) \wedge (\texttt{satB}(\beta, \psi) = \mathit{True}) \\ \bot & (\texttt{satB}(\beta, \phi) = \bot) \vee (\texttt{satB}(\beta, \psi) = \bot) \\ \mathit{False} & \text{otherwise} \end{cases}$$

$$\texttt{satB}(\beta, \neg\phi) = \begin{cases} \mathit{True} & \texttt{satB}(\beta, \phi) = \mathit{False} \\ \bot & \texttt{satB}(\beta, \phi) = \bot \\ \mathit{False} & \texttt{satB}(\beta, \phi) = \mathit{True} \end{cases}$$

$$\texttt{satB}(\beta, Q \triangleright \phi) = \texttt{satB}(\beta \parallel Q, \phi)$$

$$\texttt{satB}(\beta, \phi \mathbf{U}_{[a,b]} \psi) = \begin{cases} (\texttt{signalT}(\texttt{tube}(\beta, |\phi \mathbf{U}_{[a,b]} \psi|, \phi \mathbf{U}_{[a,b]} \psi)))(0) & r(\beta) \leqslant \theta \\ \bot & \text{otherwise} \end{cases}$$

For atomic propositions, logical combinations, and the context modality, we need only determine whether a ball lies entirely inside or outside the region defined by the formula, with some refinement for combinations with the mixed value $\bot$. For temporal formulae, if the ball has a radius too large for reliable extrapolation by the expansion function, then we return $\bot$. Note, though, that where `satB` has been called from within the loop of `contextSignal` this will immediately trigger bisection and further calls to `satB` over a smaller region. Finally, to compute validity of a temporal formula over a ball of radius θ or less, we compute the signal over a flow tube trajectory and take its initial value.

That, of course, requires that we compute a signal over the tube trace.

Definition 9.2.7 (`signalT`)**.** The function $\texttt{signalT} : \mathit{Tube} \times \Phi \to \mathit{Signal}_{\bot}$ is

defined recursively as follows:

$$\begin{aligned}
\texttt{signalT}(T, \mathit{Atom}) &= \texttt{basicSignalT}(T, \mathit{Atom}) \\
\texttt{signalT}(T, \phi \wedge \psi) &= \texttt{signalT}(T, \phi) \wedge \texttt{signalT}(T, \psi) \\
\texttt{signalT}(T, \neg\phi) &= \neg(\texttt{signalT}(T, \phi)) \\
\texttt{signalT}(T, \phi \mathbf{U}_{[a,b]} \psi) &= \texttt{signalT}(T, \phi) \mathbf{U}_{[a,b]} \texttt{signalT}(T, \psi) \\
\texttt{signalT}(T, Q \triangleright \phi) &= \texttt{contextSignalT}(T, Q, \phi) \ .
\end{aligned}$$

This uses the operations on three-valued signals from Definition 9.2.3 and auxiliary functions `basicSignalT` and `contextSignalT`.

`basicSignalT` : $\mathit{Tube} \times \mathit{Atomic} \to \mathit{Signal}_\perp$

For this $\texttt{basicSignalT}(T, \mathit{Atom})$ is the finite three-valued signal s where for each $(t_i, \beta_i) \in T$ we have

$$s([t_i, t_i + \rho)) = \texttt{satB}(\beta_i, \mathit{Atom}) \ .$$

`contextSignalT` : $\mathit{Tube} \times \mathbb{P} \times \Phi \to \mathit{Signal}_\perp$

Function $\texttt{contextSignalT}(T, Q, \phi)$ computes a three-valued signal as follows. Let W be the set of all balls in the tube trace T, let β_W be the minimum bounding ball around W, and let I be the interval $[t_0, t_n + \rho)$ including time points $t_0, \ldots, t_n$ of T. If $\texttt{satB}(\beta_W \parallel Q, \phi)$ is either *True* or *False* then we have our signal defined by

$$s(I) = \texttt{satB}(\beta_W \parallel Q, \phi) \ .$$

Otherwise, if the tube trace T contains only a single time point (t, β) then

$$s(I) = \texttt{satB}(\beta \parallel Q, \phi) \ .$$

Finally, if `satB` gives $\perp$ and T contains multiple time points, then bisect T and repeat the procedure for each new tube T, set of balls W and interval I.

Computation of a three-valued signal for a flow tube is very similar to that for the boolean signal of a trajectory, except that we deal in balls rather than points. However, where in `contextSignal` we group points into a ball to send to `satB`, here we group balls into their bounding ball, and still need only call `satB`—closing off the mutually recursive definition of our four functions.

9.3 Conclusions

The key advance here is that by using sensitivity calculations to drive the expansion function, we may replace multiple calls to a numerical solver to repeatedly compute traces with a single call to compute a flow tube. The full power of this is engaged when we have alternation of temporal and contextual modalities, and must therefore calculate trajectories starting at many points along a curve. The sensitive model checker groups trajectories starting within θ of each other together into a single flow tube and a single signal. In the best case, we may tremendously reduce the number of calls to the numerical solver and the number of signals to compute; in the worst case, subdivision leads us to the previous algorithm of a trace at every point, albeit after an additional amount of work.

The idea also begins to address the problem of intermediate values between discrete points on a trace. When `satB` evaluates a formula of $\mathcal{LBC}$ across a ball including several points of a trajectory, it also does so for all values between them. This extends to flow tubes, too, surrounding all trajectories that start in their initial ball. This considerably extends the earlier, purely pointwise, algorithm. However, it is not necessarily complete as a very volatile trajectory may conceivably range outside this enclosing ball in between trace points.

Sensitive model checking for $\mathcal{LBC}$ is so far only a proposed method—there is not yet any implementation or testing of the method. We propose the continuation of this investigation as future work, especially the implementation and testing of the method, given that the goal is practical in nature.

Chapter 10

Conclusions

In this chapter we draw the thesis to a conclusion. We begin by summarising the main contributions of the thesis in Section 10.1, in Section 10.2 we evaluate the relative merit of these contributions and to what extent they have advanced the field, and in Section 10.3 we present some ideas for further work and the future development of $\mathcal{LBC}$.

10.1 Summary

In summary, the main contributions of this thesis are:

- the $\mathcal{LBC}$ language definition and formal semantics;
- a number of model-checking algorithms for $\mathcal{LBC}$, the analysis of their complexity, and the comparison of their runtime performance;
- an interdisciplinary case-study on the analysis of posttranslational biochemical oscillator models using $\mathcal{LBC}$;
- and implementations, a systematic testing and profiling suite for the algorithms, and a user interface implementation.

We have shown, in $\mathcal{LBC}$, that the definition of a temporal logic equipped with both real-valued constraints and a context modality is possible and indeed tractable. This gives us the means to concisely and precisely express properties of biochemical (or indeed other) processes which involve temporal behaviour and some

environmental context, external influence, or the introduction of new species or agents. We have seen how this expressiveness extends to the concise, declarative definition of *computational experiments*.

Our collection of algorithms show that improvements and optimisations can be made with respect to the tractability of model checking and that there is room for further improvement. We have also shown that whilst the complexity of model checking increases with the *sandwich alternation depth* of formulae, this complexity is not too great for reasonable practical applications and can possibly be managed further by harnessing tools from sensitivity analysis.

We have shown that a number of interesting and useful properties can be expressed in $\mathcal{LBC}$. In the case study we have shown that not only is it possible to arbitrarily express interesting properties, but it is possible to express highly useful properties of real models in the biochemistry domain, with practical application in the analysis of biochemical models. We have shown that statements in $\mathcal{LBC}$ can be thought of as expressing *computational experiments* which can be performed automatically by means of the model checker. Indeed, many of these computational experiments can be *higher-order* meaning that one succinct and precise specification in $\mathcal{LBC}$ can represent a number of experiments all of which can be automatically executed at once by the model checker.

We have also shown that the tools required to support these processes can be implemented efficiently and can be used to perform a real analysis of biochemical models.

10.2 Evaluation

There are many cases in the analysis of biochemical models where we wish to understand the behaviour of a model or system in the presence of another model or system. In this thesis we have seen some specific examples of this: behaviour in the presence of inhibitors, behaviour when the dynamics of two models are coupled, and phase response characteristics.

We have shown in this thesis that it is possible to define a formal specification language, $\mathcal{LBC}$, for these sorts of properties and there exists a tractable solution to implementing such a language. We have shown that $\mathcal{LBC}$ can be used to express

and automatically verify complex specifications of this sort. We go further than this and show that $\mathcal{LBC}$ has the ability to express higher-order computational experiments and, by using the associated computational tools, these experiments can be executed and verified automatically and tractably.

Despite the fact that for a good selection of these properties, or experiments, the verification can be computed in reasonable time, the complexity of checking $\mathcal{LBC}$ increases rapidly in the sandwich alternation depth of the formula and thus can become very costly very quickly. We have shown however for many practical applications the computation time is quite reasonable. Moreover, this situation may possibly be improved by using sensitivity analysis to compute flows or groups of trajectories, instead of the arbitrarily large number of discrete trajectories the existing algorithms require to compute a temporally nested context modality.

The use of $\mathcal{LBC}$ as a platform for declaring and executing computational experiments is clearly beneficial and a great improvement over the existing approach of manually defining and implementing a set of computational—or indeed laboratory—experiments.

Thus, the motivations for this thesis, as set out in Section 1.1, have all been satisfied: the definition of the language, tractable implementation, and the development of a logic for cπ which harnesses the power of compositionality.

However, the usefulness of $\mathcal{LBC}$ is not unhampered by problems. For instance, we have seen the effect of computational complexity, but what of intellectual complexity? Spatio-temporal logic is hardly a trivial way to encode complex properties and, indeed, as the properties become more complex so does the formula which describes them. When we consider that we wish this to be a practical language, useful for biological modellers, we would be misguided in thinking that a language like $\mathcal{LBC}$ would be easy to learn by anyone who does not have a grounding in computational or mathematical logic.

Moreover, our modelling language cπ suffers from the same problem. It is far from being an intuitive language for the expression of biochemical networks; its terse, algebraic syntax and its use of channel passing for complex formation would undoubtedly seem arcane to a biologist. Indeed, as discussed in Chapter 2, there are a number of languages which have a syntax which is much more aligned to a biologists view of a biochemical network.

In defence of these points, though, both $\mathcal{LBC}$ and cπ are to be considered foundational languages. They are proofs-of-concept that show that such structures and properties can be expressed in a computable language. One could conceivably define a more "user-friendly" language which compiles down to these foundational languages. Such a language should be more closely aligned to the cognitive model which a biologist has for biochemical networks.

We should also repeat the point that $\mathcal{LBC}$ could be re-defined for a different underlying modelling language. The language would need to have some notion of process composition and some notion to the value of agents (e.g. number of agents or concentration), but if a language has these properties then $\mathcal{LBC}$ could be defined over them.

10.3 Further work

There are number of avenues for further work and ample opportunity for developing the field of spatio-temporal logics for formal verification and systems analysis. In particular there are two strands of unfinished work which follow directly from this thesis.

The first strand of work is the continuation of the work, from Chapter 9, on using sensitivity analysis to reduce the cost associated with numerical solving for nested context modalities. At the time of writing the technique has not been implemented in the tool suite and, therefore, no experimental results have been produced to determine the real cost of the technique for practical examples.

The second strand of work is the use of event detection or root finding techniques for the calculation of Boolean signals. The construction of signals using event detection would negate the need to compute a whole simulation trace and the need to then convert that trace to a Boolean signal. The basic signals could be calculated directly by finding the times at which the atomic propositions of a formula change their value, the signal combinators would remain the same. Likewise, the context modality signal would only require the ODEs to be re-calculated (not solved) and event detection done for its sub-formula.

An alternative direction which could be pursued might be to investigate *robustness* and *degree of satisfaction* measures for the logic. These measures have

been defined for temporal logic by Fages and Rizk [47, 84] and for signal temporal logic by Donzé and Maler [43]. Such measures can be useful for parameter estimation [83] and for robustness analysis of process models [40, 49]. These extensions to the logic would increase the degree to which the potential analysis is quantitative, that is, not only would the logic express quantitative properties, but the result of model checking would also be quantitative; they, in effect, give a measure of how well satisfied or far away from satisfaction a formula is.

We also propose that further investigation goes into the use of both $\mathcal{LBC}$ and cπ for applications other than biochemistry. It is feasible that cπ can be used for modelling computer systems with large numbers of agents, for example large peer-to-peer distributed systems; the continuous approximation of the underlying stochastic behaviour is valid, given that agents are in large populations—as we see in chemical agents, and useful for improving computation time. For these sorts of systems it would be possible to use $\mathcal{LBC}$ for expressing behaviour when more than one such system needs to work together or when some affecting system is introduced. For example, in a model of a large peer-to-peer network (e.g. [16]), $\mathcal{LBC}$ could be used to analyse the security threat of another network of agents which have some adverse affect on the network.

Finally, the direction that was discussed in the previous section is a very worthwhile avenue for development. The development of more "user-friendly"—or indeed "biologist-friendly"—languages which either have their own semantics or compile down to $\mathcal{LBC}$ and cπ are important for the practical uptake of formal modelling languages and specification languages in fields other than computer science.

Appendix A

Profiling model and parameters

The model used for producing the experimental results in Sections 5.3 and 6.2 is equivalent to the model described in Section 3.3:

$$S \triangleq (\nu\{x \overset{r_u}{-} r, x \overset{r_r}{-} r\})s\,\langle x\rangle\,.(u.S + r.P)$$
$$E \triangleq e(x).x.E$$
$$P \triangleq \tau_{r_d}.0$$

$$\Pi \triangleq c_S \cdot S \parallel c_E \cdot E \parallel c_P \cdot P$$
$$N_\Pi = \{s \overset{r_b}{-} e\}$$

along with the process Q which acts and an inhibitor and is used in testing the context modality:

$$Q \triangleq c_I \cdot I \qquad\qquad I \triangleq (\nu\{x \overset{r_j}{-} u\})i\,\langle x\rangle\,.u.I$$
$$N_Q = \{i \overset{r_i}{-} e\}$$

The parameters are as follows:

$$r_d = 0.5 \qquad r_b = 1.0 \qquad c_S = 1.0 \qquad c_E = 0.5$$
$$r_u = 0.5 \qquad r_r = 1.0 \qquad c_P = 0.0 \qquad c_I = 0.5$$
$$r_i = 2.0 \qquad r_j = 0.1$$

The propositions used in the test formulae are defined as follows:

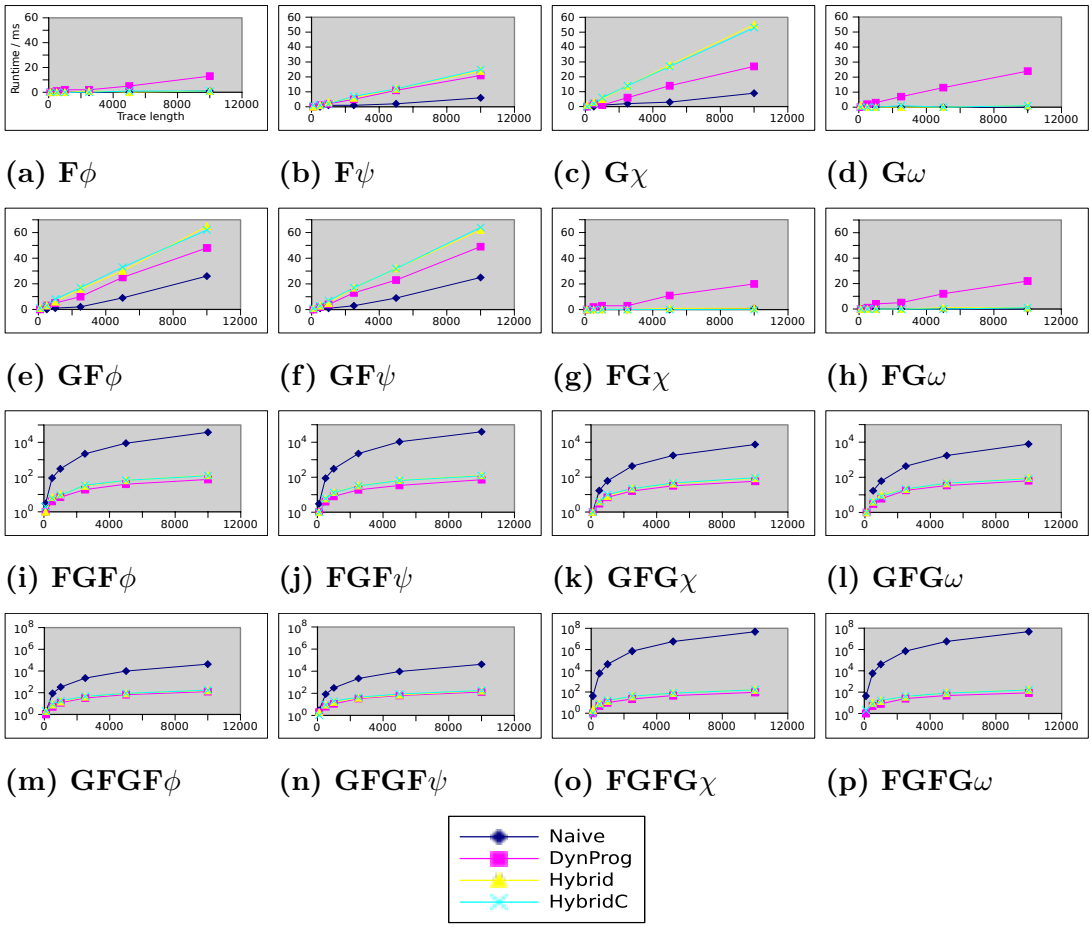
$$\phi = [P] > 0.05 \qquad\qquad \psi = [S] \leqslant 0.01$$
$$\chi = [E] > 0.1 \qquad\qquad \omega = [E] > 0.4$$

Appendix B

Systematic profiling results

This appendix contains the extended systematic profiling results from Chapters 5 and 6.

Figure B.1 Systematic performance results, for the trace-based model checker, for the temporal fragment of $\mathcal{LBC}$. ϕ is true towards the start of the trace, ψ is true only towards the end of the trace, χ is only false towards the end of the trace, and ω is false towards the start of the trace. Note that plots (i)-(p) use a log scale for runtime.



(a) $\mathbf{F}\phi$ **(b)** $\mathbf{F}\psi$ **(c)** $\mathbf{G}\chi$ **(d)** $\mathbf{G}\omega$

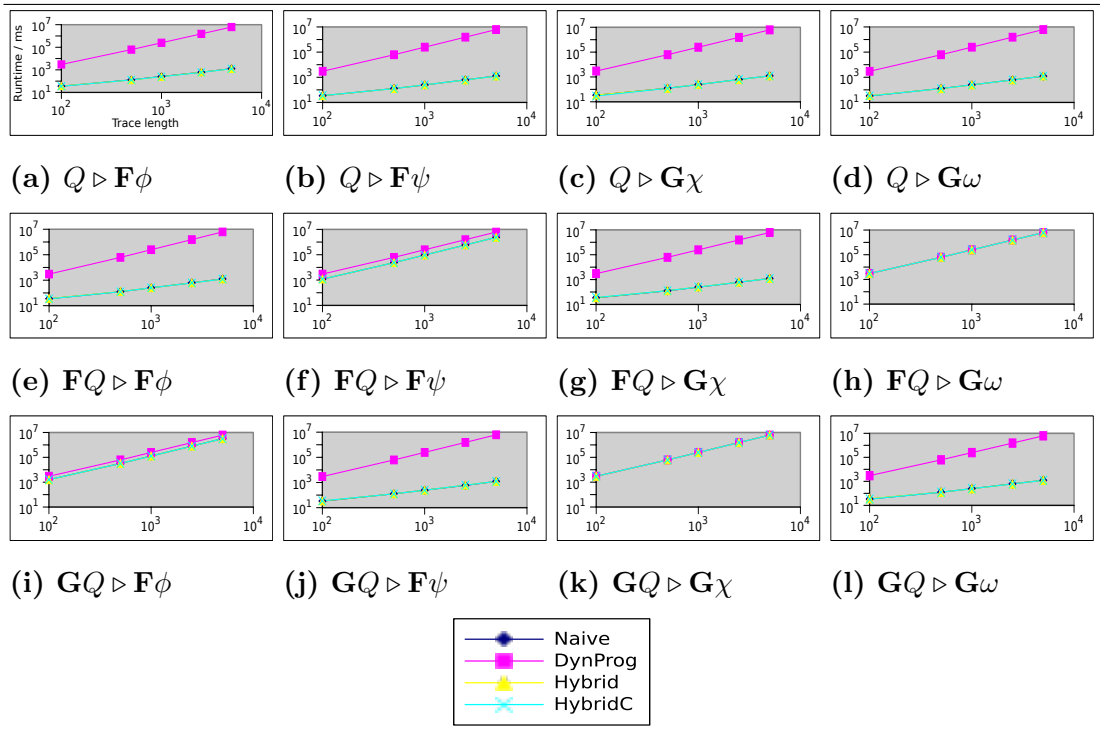
(e) $\mathbf{GF}\phi$ **(f)** $\mathbf{GF}\psi$ **(g)** $\mathbf{FG}\chi$ **(h)** $\mathbf{FG}\omega$

(i) $\mathbf{FGF}\phi$ **(j)** $\mathbf{FGF}\psi$ **(k)** $\mathbf{GFG}\chi$ **(l)** $\mathbf{GFG}\omega$

(m) $\mathbf{GFGF}\phi$ **(n)** $\mathbf{GFGF}\psi$ **(o)** $\mathbf{FGFG}\chi$ **(p)** $\mathbf{FGFG}\omega$

(q) Legend

Figure B.2 Systematic performance results, for the trace-based model checker, for the full logic $\mathcal{LBC}$. Note that these plots have a log-log scale. ϕ, ψ, χ, ω are as in figure B.1.



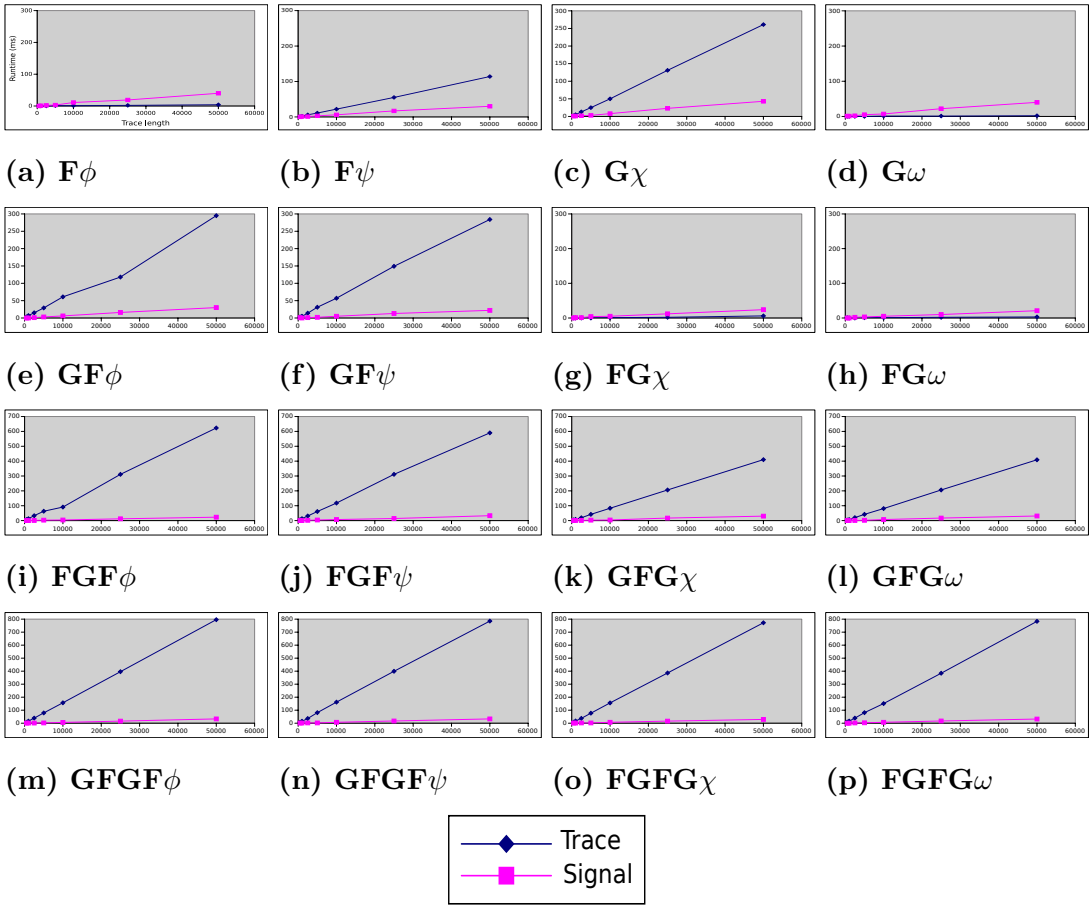
(a) $Q \triangleright \mathbf{F}\phi$ **(b)** $Q \triangleright \mathbf{F}\psi$ **(c)** $Q \triangleright \mathbf{G}\chi$ **(d)** $Q \triangleright \mathbf{G}\omega$

(e) $\mathbf{F}Q \triangleright \mathbf{F}\phi$ **(f)** $\mathbf{F}Q \triangleright \mathbf{F}\psi$ **(g)** $\mathbf{F}Q \triangleright \mathbf{G}\chi$ **(h)** $\mathbf{F}Q \triangleright \mathbf{G}\omega$

(i) $\mathbf{G}Q \triangleright \mathbf{F}\phi$ **(j)** $\mathbf{G}Q \triangleright \mathbf{F}\psi$ **(k)** $\mathbf{G}Q \triangleright \mathbf{G}\chi$ **(l)** $\mathbf{G}Q \triangleright \mathbf{G}\omega$

(m) Legend

Figure B.3 Systematic performance results, for the signal-based model checker, for the temporal fragment of $\mathcal{LBC}$. ϕ is true towards the start of the trace, ψ is true only towards the end of the trace, χ is only false towards the end of the trace, and ω is false towards the start of the trace.



(a) $\mathbf{F}\phi$ **(b)** $\mathbf{F}\psi$ **(c)** $\mathbf{G}\chi$ **(d)** $\mathbf{G}\omega$

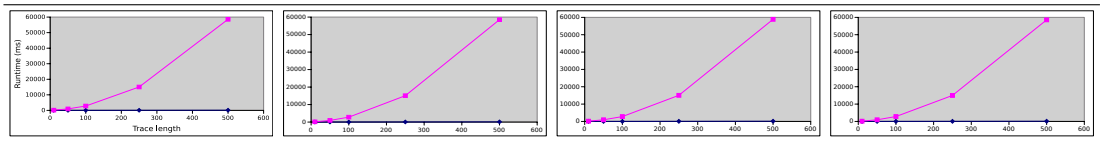
(e) $\mathbf{GF}\phi$ **(f)** $\mathbf{GF}\psi$ **(g)** $\mathbf{FG}\chi$ **(h)** $\mathbf{FG}\omega$

(i) $\mathbf{FGF}\phi$ **(j)** $\mathbf{FGF}\psi$ **(k)** $\mathbf{GFG}\chi$ **(l)** $\mathbf{GFG}\omega$

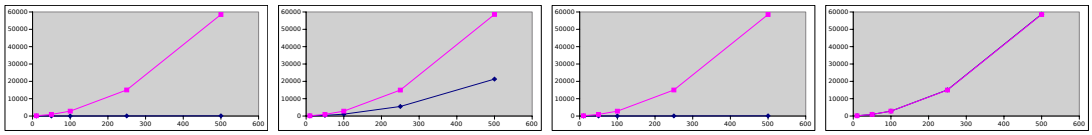
(m) $\mathbf{GFGF}\phi$ **(n)** $\mathbf{GFGF}\psi$ **(o)** $\mathbf{FGFG}\chi$ **(p)** $\mathbf{FGFG}\omega$

(q) Legend

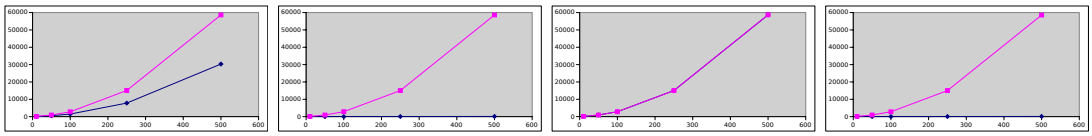
Figure B.4 Systematic performance results, for the signal-based model checker, for the full logic $\mathcal{LBC}$. ϕ, ψ, χ, ω are as in figure B.3.



(a) $Q \triangleright \mathbf{F}\phi$ **(b)** $Q \triangleright \mathbf{F}\psi$ **(c)** $Q \triangleright \mathbf{G}\chi$ **(d)** $Q \triangleright \mathbf{G}\omega$



(e) $\mathbf{F}Q \triangleright \mathbf{F}\phi$ **(f)** $\mathbf{F}Q \triangleright \mathbf{F}\psi$ **(g)** $\mathbf{F}Q \triangleright \mathbf{G}\chi$ **(h)** $\mathbf{F}Q \triangleright \mathbf{G}\omega$



(i) $\mathbf{G}Q \triangleright \mathbf{F}\phi$ **(j)** $\mathbf{G}Q \triangleright \mathbf{F}\psi$ **(k)** $\mathbf{G}Q \triangleright \mathbf{G}\chi$ **(l)** $\mathbf{G}Q \triangleright \mathbf{G}\omega$

(m) Legend

Appendix C

Case study model and parameters

This appendix details the model and parameters used in the case study in Chapter 7.

C.1 Basic Jolley model

The basic Jolley PTO model is constructed in cπ as follows:

Enzymes

$$\begin{aligned} E &\triangleq e(x).x.E \\ F &\triangleq f(x).x.F \end{aligned}$$

Substrate

$$\begin{aligned} S00 &\triangleq (\nu M_{00})\ s00a\langle be\rangle.(u.S00 + ra.S01) \\ &\qquad + s00b\langle be\rangle.(u.S00 + rb.S10) \\ S01 &\triangleq (\nu M_{01})\ s01e\langle be\rangle.(u.S01 + r.S11) \\ &\qquad + s01f\langle bf\rangle.(u.S01 + r.S00) \\ S10 &\triangleq (\nu M_{10})\ s10e\langle be\rangle.(u.S10 + r.S11) \\ &\qquad + s10f\langle bf\rangle.(u.S10 + r.S00) \\ S11 &\triangleq (\nu M_{11})\ s11a\langle bf\rangle.(u.S11 + ra.S01) \\ &\qquad + s11b\langle bf\rangle.(u.S11 + rb.S10) \end{aligned}$$

Process

$$\Pi \triangleq c_S \cdot S00 \parallel c_E \cdot E \parallel c_F \cdot F$$

where

$$c_S = 10^5, c_E = 1, c_F = 1.$$

Affinity nets with rates for Cluster 1

$$M_{00} = \{be \leftrightarrow u : 10.02,\ be \leftrightarrow ra : 163.31,\ be \leftrightarrow rb : 0\}$$

$$M_{10} = \{be \leftrightarrow u : 10.02,\ be \leftrightarrow r : 8.17,\ bf \leftrightarrow u : 10.02,\ bf \leftrightarrow r : 40.83\}$$

$$M_{01} = \{be \leftrightarrow u : 10.02,\ be \leftrightarrow r : 40.83,\ bf \leftrightarrow u : 10.02,\ bf \leftrightarrow r : 8.17\}$$

$$M_{11} = \{bf \leftrightarrow u : 10.02,\ bf \leftrightarrow ra : 0,\ bf \leftrightarrow rb : 163.31\}$$

$$M = \{s00a \leftrightarrow e : 818.18,\ s00b \leftrightarrow e : 0,\ s01e \leftrightarrow e : 13.64,\ s10e \leftrightarrow e : 4903.17,\ s01f \leftrightarrow f : 4903.17,\ s10f \leftrightarrow f : 13.64,\ s11a \leftrightarrow f : 0,\ s11b \leftrightarrow f : 818.18\}$$

Affinity nets with rates for Cluster 2

$$M_{00} = \{be \leftrightarrow u : 10.02,\ be \leftrightarrow ra : 14.12,\ be \leftrightarrow rb : 7.06\}$$

$$M_{10} = \{be \leftrightarrow u : 10.02,\ be \leftrightarrow r : 0,\ bf \leftrightarrow u : 10.02,\ bf \leftrightarrow r : 105.91\}$$

$$M_{01} = \{be \leftrightarrow u : 10.02,\ be \leftrightarrow r : 105.91,\ bf \leftrightarrow u : 10.02,\ bf \leftrightarrow r : 0\}$$

$$M_{11} = \{bf \leftrightarrow u : 10.02,\ bf \leftrightarrow ra : 7.06,\ bf \leftrightarrow rb : 14.12\}$$

$$M = \{s00a \leftrightarrow e : 7074.12,\ s00b \leftrightarrow e : 70.74,\ s01e \leftrightarrow e : 2.12,\ s10e \leftrightarrow e : 0,\ s01f \leftrightarrow f : 0,\ s10f \leftrightarrow f : 2.12,\ s11a \leftrightarrow f : 70.74,\ s11b \leftrightarrow f : 7074.12\}.$$

C.2 Coupled jPTOs model

The coupled model is constructed from the same substrate and enzyme species as the basic model in Section C.1. The second jPTO is a copy of the original substrate, renamed so it forms a distinct species:

$$\begin{aligned}
T00 &\triangleq (\nu M_{00})\ t00a\langle be\rangle.(u.T00 + ra.T01) \\
&\qquad + t00b\langle be\rangle.(u.T00 + rb.T10) \\
T01 &\triangleq (\nu M_{01})\ t01e\langle be\rangle.(u.T01 + r.T11) \\
&\qquad + t01f\langle bf\rangle.(u.T01 + r.T00) \\
T10 &\triangleq (\nu M_{10})\ t10e\langle be\rangle.(u.T10 + r.T11) \\
&\qquad + t10f\langle bf\rangle.(u.T10 + r.T00) \\
T11 &\triangleq (\nu M_{11})\ t11a\langle bf\rangle.(u.T11 + ra.T01) \\
&\qquad + t11b\langle bf\rangle.(u.T11 + rb.T10)
\end{aligned}$$

The process term is the same as above, but with the addition of the new (copy) substrate:

$$\Pi \triangleq c_S \cdot S00 \parallel c_T \cdot T00 \parallel c_E \cdot E \parallel c_F \cdot F$$

where

$$c_S = 10^5, c_T = 10^5, c_E = 1, c_F = 1,$$

and the global affinity net is then extended to allow the new substrate to interact with the enzymes:

$$\begin{aligned}
M = \{ & s00a \leftrightarrow e : 818.18, & & t00a \leftrightarrow e : 181.18, \\
& s00b \leftrightarrow e : 0, & & t00b \leftrightarrow e : 0, \\
& s01e \leftrightarrow e : 13.64, & & t01e \leftrightarrow e : 13.64, \\
& s10e \leftrightarrow e : 4093.17, & & t10e \leftrightarrow e : 4093.17, \\
& s01f \leftrightarrow f : 4093.17, & & t01f \leftrightarrow f : 4093.17, \\
& s10f \leftrightarrow f : 13.64, & & t10f \leftrightarrow f : 13.64, \\
& s11a \leftrightarrow f : 0, & & t11a \leftrightarrow f : 0, \\
& s11b \leftrightarrow f : 818.18, & & t11b \leftrightarrow f : 818.18\}.
\end{aligned}$$

C.3 Weaker coupled jPTOs

For the weaker coupled model we have a separate phosphatase for each substrate. The model in Section C.2 is extended by replacing species F with the following:

$$\begin{aligned} F_S &\triangleq fs(x).x.F_S \\ F_T &\triangleq ft(x).x.F_S \end{aligned}$$

and the process term is extended:

$$\Pi \triangleq c_S \cdot S00 \parallel c_T \cdot T00 \parallel c_E \cdot E \parallel c_{F_S} \cdot F_S \parallel c_{F_T} \cdot F_T$$

where

$$c_S = 10^5, c_T = 10^5, c_E = 1, c_{F_S} = 1, c_{F_T} = 1,$$

and the affinity net is altered so each substrate only has affinity for one of the phosphatases:

$$\begin{aligned} M = \{&s00a \leftrightarrow e : 818.18, \\ &s00b \leftrightarrow e : 0, \\ &s01e \leftrightarrow e : 13.64, \\ &s10e \leftrightarrow e : 4093.17, \\ &s01f \leftrightarrow fs : 4093.17, \\ &s10f \leftrightarrow fs : 13.64, \\ &s11a \leftrightarrow fs : 0, \\ &s11b \leftrightarrow fs : 818.18, \\ &t00a \leftrightarrow e : 181.18, \\ &t00b \leftrightarrow e : 0, \\ &t01e \leftrightarrow e : 13.64, \\ &t10e \leftrightarrow e : 4093.17, \\ &t01f \leftrightarrow ft : 4093.17, \\ &t10f \leftrightarrow ft : 13.64, \\ &t11a \leftrightarrow ft : 0, \\ &t11b \leftrightarrow ft : 818.18\}. \end{aligned}$$

C.4 Coupling non-identical jPTOs

To couple non-identical jPTOs we give one substrate the parameters for Cluster 1 and the other the parameters for Cluster 2. The model is constructed as in Section C.2, but with the following global affinity network:

$$
\begin{aligned}
M = \{&\mathit{s00a} \leftrightarrow e : 818.18, \\
&\mathit{s00b} \leftrightarrow e : 0, \\
&\mathit{s01e} \leftrightarrow e : 13.64, \\
&\mathit{s10e} \leftrightarrow e : 4093.17, \\
&\mathit{s01f} \leftrightarrow f : 4093.17, \\
&\mathit{s10f} \leftrightarrow f : 13.64, \\
&\mathit{s11a} \leftrightarrow f : 0, \\
&\mathit{s11b} \leftrightarrow f : 818.18, \\
&\mathit{t00a} \leftrightarrow e : 7074.12, \\
&\mathit{t00b} \leftrightarrow e : 70.74, \\
&\mathit{t01e} \leftrightarrow e : 2.12, \\
&\mathit{t10e} \leftrightarrow e : 0, \\
&\mathit{t01f} \leftrightarrow f : 0, \\
&\mathit{t10f} \leftrightarrow f : 2.12, \\
&\mathit{t11a} \leftrightarrow f : 70.74, \\
&\mathit{t11b} \leftrightarrow f : 7074.12\}.
\end{aligned}
$$

C.5 Coupling out of phase

Coupling out of phase merely requires a change of initial conditions. To construct a process with initial conditions representing the concentrations a specific time in the evolution of a model one can use the `evolve` function in the toolset. To construct such a process by hand is infeasible.

Models for out-of-phase coupling were constructed in this way, corresponding to the model evolved for half a phase and a quarter of a phase.

C.6 Driving other reactions

To construct the model which drives another phosphorylation reaction, we first construct P which is the molecule to be phosphorylated:

$$
\begin{aligned}
P &\triangleq (\nu M_P)\ p\langle x\rangle.(u.P + r.P') \\
P' &\triangleq \tau_d.P
\end{aligned}
$$

where $d = 10^{-4}$ and $M_P = \{x \leftrightarrow u : 1, x \leftrightarrow r : 1\}$.

The model is then the same as the basic model in Section C.1, but with a new site, which interacts with the P molecule, added to the *S11* state of the substrate:

$$\begin{aligned} S11 \triangleq\ & (\nu M_{11})\ s11a\langle bf\rangle.(u.S11 + ra.S01) \\ & + s11b\langle bf\rangle.(u.S11 + rb.S10) \\ & + s11p(x).x.S11 \end{aligned}$$

the new molecule added to the process:

$$\Pi \triangleq c_S \cdot S00 \parallel c_E \cdot E \parallel c_F \cdot F \parallel c_P \cdot P$$

where

$$c_S = 10^5, c_E = 1, c_F = 1, c_P = 10^5$$

and the affinity net is extended with

$$\begin{aligned} M = \{ & s00a \leftrightarrow e : 818.18, \\ & s00b \leftrightarrow e : 0, \\ & s01e \leftrightarrow e : 13.64, \\ & s10e \leftrightarrow e : 4903.17, \\ & s01f \leftrightarrow f : 4903.17, \\ & s10f \leftrightarrow f : 13.64, \\ & s11a \leftrightarrow f : 0, \\ & s11b \leftrightarrow f : 818.18, \\ & s11p \leftrightarrow p : 3 \times 10^{-4}\}. \end{aligned}$$

C.7 Perturbation

To construct the model with a pulse of inhibitor, we take the model in Section C.6 and replace the driven species P with an inhibitor *In* which decays and a species *ProdIn* which autonomously produces the inhibitor:

$$\begin{aligned} In &\triangleq (\nu M_{In})\ p\langle x\rangle u.In + \tau_d.0 \\ ProdIn &\triangleq \tau_d.P \end{aligned}$$

where $M_{In} = \{x \leftrightarrow u : 0.1\}$ and $d = 5 \times 10^{-3}$ and the inhibitor producer added to the process:

$$\Pi \triangleq c_S \cdot S00 \parallel c_E \cdot E \parallel c_F \cdot F \parallel c_P \cdot ProdIn$$

where

$$c_S = 10^5, c_E = 1, c_F = 1, c_P = 10^5$$

In this model the inhibitor binds to the substrate in its $S11$ state. The models where the inhibitor binds to one or the other of the enzymes is constructed in a similar way, with a corresponding new site on the enzyme instead of the substrate. When binding to the enzyme, however the rate should be adjusted from 3×10^{-4} to 5.

Bibliography

[1] Abraham, U., Granada, A. E., Westermark, P. O., Heine, M., Kramer, A., and Herzel, H. Coupling governs entrainment range of circadian clocks. *Molecular systems biology 6*, 1 (Nov. 2010).

[2] Aiello, M., and Benthem, J. V. A Modal Walk Through Space. *Journal of Applied Non-Classical Logics 12* (2002), 319–363.

[3] Akman, O., Ciocchetta, F., Degasperi, A., and Guerriero, M. Modelling biological clocks with Bio-PEPA: stochasticity and robustness for the Neurospora Crassa circadian network. *Lecture Notes In Computer Science 5688* (2009), 52–67.

[4] Alur, R., Feder, T., and Henzinger, T. The benefits of relaxing punctuality. *Journal of the ACM (JACM) 43*, 1 (1996), 116–146.

[5] Alur, R., and Henzinger, T. Logics and models of real time: A survey. *Lecture Notes In Computer Science 600* (1992), 74–106.

[6] Antoniotti, M., Policriti, A., Ugel, N., and Mishra, B. Model building and model checking for biochemical processes. *Cell biochemistry and biophysics 38*, 3 (Jan. 2003), 271–86.

[7] Asarin, E., Donzé, A., Maler, O., and Nickovic, D. Parametric identification of temporal properties. *Lecture Notes In Computer Science 7186* (2012), 147–160.

[8] Baeten, J. C. M. A brief history of process algebra. *Theoretical Computer Science 335*, 2-3 (2005), 131–146.

[9] Ballarini, P., and Guerriero, M. L. Query-based verification of qualitative trends and oscillations in biochemical systems. *Theoretical Computer*

Science 411, 20 (Apr. 2010), 2019–2036.

[10] Ballarini, P., Mardare, R., and Mura, I. Analysing Biochemical Oscillation through Probabilistic Model Checking. *Electronic Notes in Theoretical Computer Science 229*, 1 (Feb. 2009), 3–19.

[11] Banks, C., Clark, A., Georgoulas, A., Gilmore, S., Hilston, J., Milios, D., and Stark, I. Stochastic modelling of the Kai-based circadian clock. *Electronic Notes in Theoretical Computer Science 296* (2013), 43–60.

[12] Banks, C. J., and Stark, I. A Logic of Behaviour in Context. *Information and Computation 236* (2014), 3–18.

[13] Bellini, P., Mattolini, R., and Nesi, P. Temporal logics for real-time system specification. *ACM Computing Surveys 32*, 1 (Mar. 2000), 12–42.

[14] Bergstra, J., and Klop, J. Process algebra for synchronous communication. *Information and control 137* (1984), 109–137.

[15] Bernot, G., Comet, J.-P., Richard, A., and Guespin, J. Application of formal methods to biological regulatory networks: extending Thomas' asynchronous logical approach with temporal logic. *Journal of Theoretical Biology 229*, 3 (Aug. 2004), 339–347.

[16] Bradley, J. T. J., Gilmore, S. T., and Hillston, J. Analysing distributed Internet worm attacks using continuous state-space approximation of process algebra models. *Journal of Computer and System Sciences 74*, 6 (Sept. 2008), 1013–1032.

[17] Buchler, N. E., and Louis, M. Molecular titration and ultrasensitivity in regulatory networks. *Journal of Molecular Biology 384*, 5 (Dec. 2008), 1106–1119.

[18] Cai, Y., Davidson, B., Ma, H., and French, C. Modeling the arsenic biosensor system. *BMC Systems Biology 1*, Suppl 1 (2007), 83.

[19] Caires, L., and Lozes, E. Elimination of quantifiers and undecidability in spatial logics for concurrency. *Theoretical computer science 358*, 2-3 (2006), 293–314.

[20] Calder, M., Duguid, A., Gilmore, S., and Hillston, J. Stronger computational modelling of signalling pathways using both continuous and

discrete-state methods. *Lecture Notes In Computer Science 4210* (2006), 63–77.

[21] Calder, M., Gilmore, S., and Hillston, J. Automatically deriving ODEs from process algebra models of signalling pathways. In *Proceedings of Computational Methods in Systems Biology* (2005), pp. 204–215.

[22] Calder, M., Gilmore, S., and Hillston, J. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. *Transactions on Computational Systems Biology VII* (2006), 1–23.

[23] Calzone, L., Chabrier-Rivier, N., Fages, F., and Soliman, S. Machine learning biochemical networks from temporal logic properties. *Transactions on Computational Systems Biology VI* (2006), 68–94.

[24] Cardelli, L. Brane calculi. *Lecture Notes In Computer Science 3082* (2005), 257–278.

[25] Cardelli, L., and Gordon, A. Anytime, anywhere: Modal logics for mobile ambients. In *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL)* (2000), ACM, pp. 365–377.

[26] Chabrier, N., and Fages, F. Symbolic model checking of biochemical networks. In *Computational Methods in Systems Biology* (2003), Springer, pp. 149–162.

[27] Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., and Schächter, V. Modeling and querying biomolecular interaction networks. *Theoretical Computer Science 325*, 1 (Sept. 2004), 25–44.

[28] Chabrier-Rivier, N., Fages, F., and Soliman, S. The biochemical abstract machine BIOCHAM. In *Computational Methods in Systems Biology* (2005), Springer, pp. 172–191.

[29] Chickarmane, V., Kholodenko, B. N., and Sauro, H. M. Oscillatory dynamics arising from competitive inhibition and multisite phosphorylation. *Journal of theoretical biology 244*, 1 (Jan. 2007), 68–76.

[30] Ciocchetta, F., and Guerriero, M. Modelling biological compartments in Bio-PEPA. *Electronic Notes in Theoretical Computer Science 227* (2009), 77–95.

[31] Ciocchetta, F., and Hillston, J. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science 410*, 33-34 (2009), 3065–3084.

[32] Clarke, E., Emerson, E., and Sifakis, J. Model checking: algorithmic verification and debugging. *Communications of the ACM 52*, 11 (2009), 74–84.

[33] Clarke, E. M., Grumberg, O., and Peled, D. A. *Model checking*. MIT Press, 1999.

[34] Danos, V., and Laneve, C. Graphs for core molecular biology. In *Computational Methods in Systems Biology* (2003), Springer, pp. 34–46.

[35] Danos, V., and Laneve, C. Formal molecular biology. *Theoretical Computer Science 325*, 1 (Sept. 2004), 69–110.

[36] de Nicola, R. D., and Loreti, M. MoMo: A modal logic for reasoning about mobility. In *Formal Methods for Components and Objects* (2004), Springer, pp. 95–119.

[37] Dematté, L., Priami, C., and Romanel, A. The BlenX Language: A Tutorial. *Lecture Notes In Computer Science 5016* (2008), 313–365.

[38] Dematté, L., Priami, C., Romanel, A., and Soyer, O. A formal and integrated framework to simulate evolution of biological pathways. In *Computational Methods in Systems Biology* (2007), Springer, pp. 106–120.

[39] Dluhoš, P., Brim, L., and Šafránek, D. On Expressing and Monitoring Oscillatory Dynamics. *Electronic Proceedings in Theoretical Computer Science 92* (Aug. 2012), 73–87.

[40] Donzé, A., Fanchon, E., Gattepaille, L. M., Maler, O., and Tracqui, P. Robustness analysis and behavior discrimination in enzymatic reaction networks. *PloS one 6*, 9 (Jan. 2011), e24246.

[41] Donzé, A., Ferrère, T., and Maler, O. Efficient Robust Monitoring for STL. *Lecture Notes In Computer Science 8044* (2013), 264–279.

[42] Donzé, A., and Maler, O. Systematic simulation using sensitivity analysis. *Lecture Notes In Computer Science 4416* (2007), 174–189.

[43] Donzé, A., and Maler, O. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems* (2010), Springer, pp. 92–106.

[44] Donzé, A., Maler, O., Bartocci, E., Nickovic, D., Grosu, R., and Smolka, S. On temporal logic and signal processing. In *10th International Symposium on Automated Technology for Verification and Analysis* (2012), Springer, pp. 92–106.

[45] Eaton, J., Bateman, D., and Hauberg, S. *GNU Octave*. Free Software Foundation, 1997.

[46] Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Meseguer, J., and Sonmez, K. Pathway logic: Symbolic analysis of biological signaling. In *Proceedings of Pacific Symposium on Biocomputing* (2002), World Scientific Pub Co Inc, pp. 400–412.

[47] Fages, F., and Rizk, a. On temporal logic constraint solving for analyzing numerical data time series. *Theoretical Computer Science 408*, 1 (Nov. 2008), 55–65.

[48] Fages, F., and Soliman, S. Formal cell biology in BIOCHAM. In *Formal Methods for Computational Systems Biology, LNCS 5016* (2008), Springer, pp. 54–80.

[49] Fainekos, G., and Pappas, G. Robustness of temporal logic specifications. In *Formal Approaches to Software Testing and Runtime Verification, LNCS 4262* (2006), Springer, pp. 178–192.

[50] Gong, H., Zuliani, P., Komuravelli, A., and Faeder, J. Computational modeling and verification of signaling pathways in cancer. In *Proceedings of the 4th International Conference on Algebraic and Numeric Biology 2010, LNCS 6479* (2012), Springer, pp. 117–135.

[51] Granada, a., Hennig, R. M., Ronacher, B., Kramer, A., and Herzel, H. Phase response curves elucidating the dynamics of coupled oscillators. *Methods in Enzymology 454* (Jan. 2009), 1–27.

[52] Guerriero, M. Qualitative and quantitative analysis of a Bio-PEPA model of the Gp130/JAK/STAT signalling pathway. *Transactions on Computational Systems Biology XI* (2009), 90–115.

[53] Hillston, J. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[54] Hoare, C. A. R. *Communicating sequential processes*. Prentice Hall, 1985.

[55] Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., and Kummer, U. COPASI—a COmplex PAthway SImulator. *Bioinformatics 22*, 24 (Dec. 2006), 3067–74.

[56] Huang, C. Y., and Ferrell, J. E. Ultrasensitivity in the mitogen-activated protein kinase cascade. *Proceedings of the National Academy of Sciences of the United States of America 93*, 19 (Sept. 1996), 10078–83.

[57] Huth, M., and Ryan, M. *Logic in computer science: modelling and reasoning about systems*. Cambridge University Press, 2004.

[58] Johnson, C. H., Mori, T., and Xu, Y. A cyanobacterial circadian clockwork. *Current Biology 18*, 17 (Sept. 2008), R816–R825.

[59] Jolley, C. C., Ode, K. L., and Ueda, H. R. A Design Principle for a Posttranslational Biochemical Oscillator. *Cell reports 2*, 4 (Oct. 2012), 938–950.

[60] Koeppl, H., Hafner, M., and Danos, V. Rule-based modeling for protein-protein interaction networks - the cyanobacterial circadian clock as a case study. In *International Workshop on Computational Systems Biology* (2009).

[61] Kurtz, T. G. Limit theorems for sequences of jump Markov processes approximating ordinary differential processes. *Journal of Applied Probability 8*, 2 (1971), 344–356.

[62] Kwiatkowski, M. *A formal computational framework for the study of molecular evolution*. PhD thesis, University of Edinburgh, 2010.

[63] Kwiatkowski, M., and Stark, I. The continuous π-calculus: A process algebra for biochemical modelling. In *Computational Methods in Systems Biology, LNCS 5307* (2008), Springer, pp. 103–122.

[64] Kwiatkowski, M., and Stark, I. On Executable Models of Molecular Evolution. *Tampere International Center for Signal Processing Series 53* (2011), 105–108.

[65] Liu, P., Kevrekidis, I. G., and Shvartsman, S. Y. Substrate-dependent control of ERK phosphorylation can lead to oscillations. *Biophysical journal 101*, 11 (Dec. 2011), 2572–81.

[66] Maler, O., and Nickovic, D. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems* (2004), Springer, pp. 152–166.

[67] Maler, O., Nickovic, D., and Pnueli, A. Checking temporal properties of discrete, timed and continuous behaviors. *Pillars of computer science, LNCS 4800* (2008), 475–505.

[68] Milner, R. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[69] Milner, R. *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[70] Milner, R. *Communicating and mobile systems: the pi-calculus*. Cambridge University Press, 1999.

[71] Monteiro, P. T., Ropers, D., Mateescu, R., Freitas, A. T., and de Jong, H. Temporal logic patterns for querying dynamic models of cellular interaction networks. *Bioinformatics 24*, 16 (Aug. 2008), i227–33.

[72] Nakajima, M., Imai, K., Ito, H., Nishiwaki, T., Murayama, Y., Iwasaki, H., Oyama, T., and Kondo, T. Reconstitution of circadian oscillation of cyanobacterial KaiC phosphorylation in vitro. *Science (New York, N.Y.) 308*, 5720 (Apr. 2005), 414–5.

[73] Peyton Jones, S. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.

[74] Pnueli, A. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (FOCS)* (1977), IEEE, pp. 45–57.

[75] Priami, C. Stochastic π-calculus. *The Computer Journal 38*, 7 (1995), 578–589.

[76] Priami, C., and Quaglia, P. Beta binders for biological interactions. In *Computational Methods in Systems Biology, LNCS 3082* (2005), Lecture Notes in Computer Science, Springer, pp. 20–33.

[77] Priami, C., Regev, A., Shapiro, E., and Silverman, W. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters 80*, 1 (2001), 25–31.

[78] Randell, D., Cui, Z., and Cohn, A. A spatial logic based on regions and connection. In *Proceedings of 3rd International Conference on Knowledge Representation and Reasoning* (1992).

[79] Regev, A., Panina, E., Silverman, W., Cardelli, L., and Shapiro, E. BioAmbients: an abstraction for biological compartments. *Theoretical Computer Science 325*, 1 (2004), 141–167.

[80] Regev, A., and Shapiro, E. Cells as computation. *Nature 419* (2002), 343.

[81] Regev, A., Silverman, W., and Shapiro, E. Representing biomolecular processes with computer process algebra: π-calculus programs of signal transduction pathways. In *Proceedings of the Pacific Symposium of Biocomputing* (2000), p. 179.

[82] Regev, A., Silverman, W., and Shapiro, E. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Proceedings of the Pacific Symposium on Biocomputing* (2001), World Scientific Pub Co Inc, pp. 459–470.

[83] Rizk, A., Batt, G., Fages, F., and Soliman, S. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *Computational Methods in Systems Biology* (2008), Springer, pp. 251–268.

[84] Rizk, A., Batt, G., Fages, F., and Soliman, S. Continuous valuations of temporal logic specifications with applications to parameter optimization and robustness measures. *Theoretical Computer Science 412*, 26 (June 2011), 2827–2839.

[85] Romanel, A. *Dynamic Biological Modelling*. PhD thesis, University of Trento, 2010.

[86] van Ooijen, G., and Millar, A. J. Non-transcriptional oscillators in circadian timekeeping. *Trends in biochemical sciences 37*, 11 (Nov. 2012), 484–92.

[87] van Zon, J. S., Lubensky, D. K., Altena, P. R. H., and ten Wolde, P. R. An allosteric model of circadian KaiC phosphorylation. *Proceedings of the National Academy of Sciences of the United States of America 104*, 18 (May 2007), 7420–5.